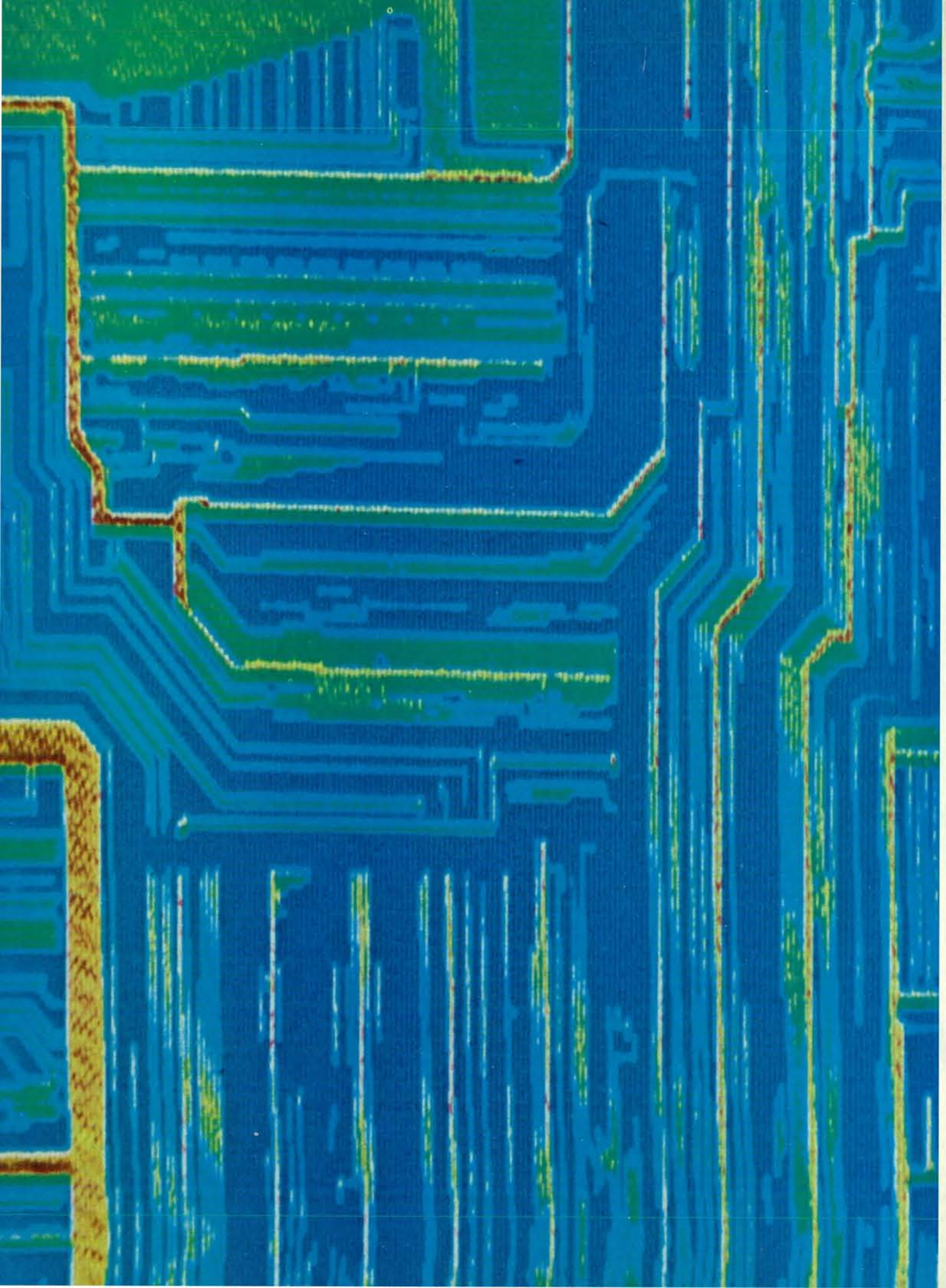


LE SCIENZE
edizione italiana di SCIENTIFIC AMERICAN

numero 195
novembre 1984
anno XVII
volume XXXIII

Il software

Più o meno come uno scultore modella la creta, il software dà forma e scopo a una macchina programmabile: i concetti e le tecniche necessari perché il calcolatore esegua gli ordini sono il tema di questo fascicolo

di Alan Kay

I calcolatori stanno all'informatica come gli strumenti musicali stanno alla musica. Il software costituisce la partitura, la cui interpretazione allarga i nostri confini ed eleva il nostro spirito. Leonardo da Vinci definiva la musica «l'arte di dare forma all'invisibile», e questa definizione è ancora più appropriata per descrivere il software. Come nel caso della musica, l'invisibilità del software non è più misteriosa di dove va a finire il grembo quando ci si alza in piedi. Il vero mistero da esplorare in questo fascicolo di «Le Scienze» è il modo in cui, data la giusta architettura, si possa fare tanto con il più semplice dei materiali.

I materiali dell'elaborazione sono i più essenziali tra i segni, immagazzinati a miliardi nell'hardware del calcolatore. In una partitura musicale la melodia è rappresentata in hardware di carta e inchiostro; in biologia il messaggio trasmesso di generazione in generazione dal DNA è contenuto nella sequenza dei nucleotidi. Proprio come per conservare i segni della scrittura vi sono molti materiali (dall'argilla al papiro, dalla pergamena alla carta e inchiostro), così per memorizzare i propri segni l'hardware dei calcolatori si è avvalso di vari sistemi fisici: alberi rotanti, perforazione di schede, flusso magnetico, tubi a vuoto, transistori e circuiti integrati stampati su chip di silicio. I segni sull'argilla o sulla carta, nel DNA e nelle memorie dei calcolatori sono ugualmente validi nella capacità di rappresentazione, ma l'unico significato intrinseco di un segno è il fatto di esserci. «L'informazione - ha notato Gregory Bateson - è qualsiasi differenza che crea una differenza.» La prima differenza è il segno; la seconda allude alla necessità di interpretazione.

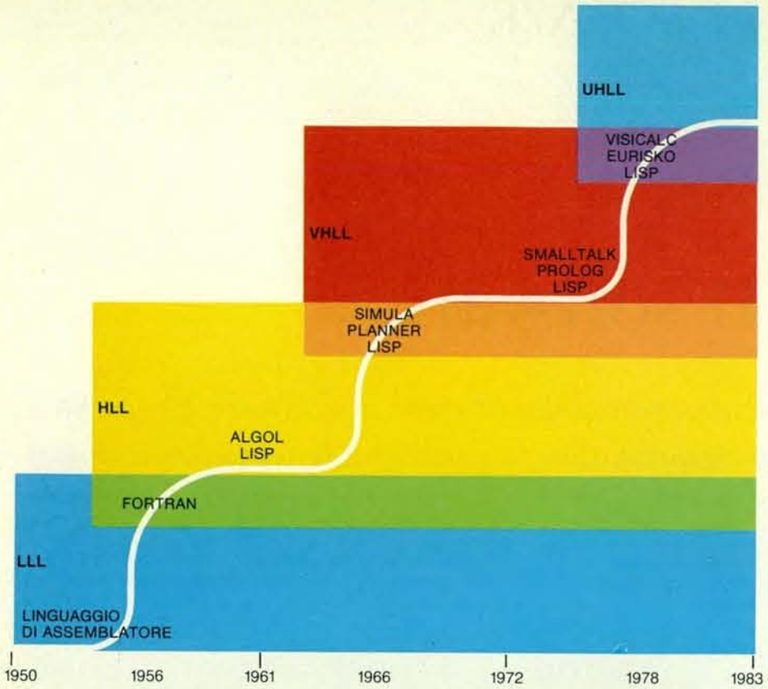
La stessa notazione che specifica la musica da grandi magazzini specifica le fughe per organo di Bach. In un calcolatore la stessa notazione può specificare tavole attuariali o dare vita a un nuovo mondo. Il fatto che la notazione per i graffiti e per i sonetti possa essere la stessa non è nuovo. Che la cosa valga anche per i calcolatori elimina gran parte del mistero che avvolge la nuova tecnologia e dà basi più solide alla nostra riflessione in materia.

Come per la maggior parte dei mezzi con cui si costruiscono le cose, si tratti di una cattedrale, di un batterio, di un sonetto, di una fuga o di un elaboratore di testi, l'architettura prevale sul materiale. Capire la creta non è capire il vaso. Che cosa è un vaso lo si può apprezzare meglio capendo i creatori e gli utenti del vaso stesso e il loro bisogno sia di dare significato al materiale sia di trarre significato dalla forma.

V'è una differenza qualitativa tra il calcolatore come mezzo d'espressione e la creta o la carta. Al pari dell'apparato genetico di una cellula vivente, il calcolatore può leggere, scrivere e seguire i propri segni fino a livelli di autointerpretazione di cui non si conoscono ancora i limiti intellettivi. Per chi voglia capire il software non si tratta quindi semplicemente di vedere il vaso invece della creta; si tratta di scorgere nei vasi lavorati al tornio da principianti (perché tutti sono principianti nella professione, ancora ai primi passi, della scienza dei calcolatori) le possibilità di là da venire della porcellana cinese o di quella di Limoges.

Non è necessario da parte mia dedicare in questa sede, ai metodi di composizione per la memorizzazione e la lettura dei segni, più tempo di quanto ne dedichi la biologia molecolare alle pro-

Il messaggio intangibile racchiuso in un mezzo materiale è l'essenza del software. Qui il messaggio è reso visibile in una immagine a contrasto di tensione: una microfotografia al microscopio elettronico a scansione di una piccola parte di un microelaboratore Intel 80186. Le caratteristiche dell'immagine sono costituite non dai conduttori e dai transistori presenti sul chip, ma dai segnali che li attraversano. La traiettoria degli elettroni secondari emessi in risposta al fascio del microscopio è influenzata dai campi elettromagnetici alla superficie del chip: le regioni di tensione più elevata attraggono gli elettroni, indebolendo il segnale che forma l'immagine. Il fascio del microscopio viene attivato solo quando il microelaboratore si trova in un particolare stato elettronico, cioè quando certi elementi logici sono «on». I colori delle righe indicano la tensione nelle linee di comunicazione metalliche che conducono a elementi logici. Là dove un segnale viaggia lungo una di queste linee si ha una regione di alta tensione. L'immagine in falsi colori è stata elaborata in modo che tali regioni, e quindi i «messaggi», siano viste in blu chiaro. Le regioni di bassa tensione sono in verde, quelle di tensione intermedia in giallo. Le righe in rosso sono conduttori a potenziale di massa, ovvero a zero volt. La microfotografia è di Timothy C. May della Intel Corporation.



I vari generi di software si sono succeduti a intervalli sporadici, come risulta dall'esempio relativo ad alcuni linguaggi di programmazione. I linguaggi sono classificati piuttosto arbitrariamente per livello, anche se i livelli *(bande in colore)* si sovrappongono. Vi sono linguaggi di basso livello (LLL), linguaggi di alto livello (HLL), linguaggi di altissimo livello (VHLL) e linguaggi di livello ultraelevato (UHLL). Nell'evoluzione dei linguaggi di programmazione a un certo punto si afferma un genere *(tratti orizzontali della linea bianca)*, a cui, dopo qualche anno, si apporta un miglioramento *(tratti curvi della linea)*. Nel tempo il linguaggio migliorato viene considerato non più solamente alla stregua di una «cosa vecchia migliore», ma di una «cosa quasi nuova» e porta al genere stabile successivo. Il linguaggio Lisp è cambiato ripetutamente, diventando ogni volta un nuovo genere.

prietà generali degli atomi. Bastano una capacità di memorizzazione dei segni sufficientemente grande e un insieme di istruzioni fra i più semplici per costruire qualsiasi ulteriore meccanismo di rappresentazione di cui si abbia bisogno, ivi compresa perfino la simulazione di tutto un nuovo calcolatore. Augusta Ada, contessa di Lovelace, il primo genio del software dei calcolatori, che programmò la macchina analitica progettata da Charles Babbage, era perfettamente consapevole dei poteri di simulazione di quella macchina di uso generale. Negli anni trenta Alan M. Turing espose il caso in modo più lucido dimostrando come un meccanismo straordinariamente semplice potesse simulare tutti i meccanismi.

L'idea che un calcolatore qualunque possa simulare qualsiasi altro calcolatore presente o futuro è importante dal punto di vista filosofico, ma non è la risposta a tutti i problemi computazionali. Troppo spesso un calcolatore semplice che pretende di essere di qualità superiore si impantana nella «melma di Turing» e non serve più a niente se i risultati sono necessari in meno di un milione di anni. In altre parole, non è da escludere l'utilità anche di miglioramenti quantitativi. Un aumento della velocità potrebbe addirittura rappresentare un miglioramento qualitativo. Si pensi alla straordinaria differenza costituita dalla possibilità di accelerare una pellicola da due a venti fotogrammi al secondo (un semplice ordine di grandezza), che porta alla percezione soggettiva del movimento continuo. Gran parte della «vita» dell'interazione visiva e uditiva dipende dal suo ritmo.

Da bambini ci siamo resi conto che si poteva dare qualsiasi forma alla creta semplicemente lavorando il materiale con entrambe le mani. Nessuno di noi, o quasi, ha imparato che qualcosa del genere vale anche per il calcolatore. Il suo materiale sembra tanto lontano dall'esperienza umana quanto un lingotto radioattivo maneggiato a distanza con pulsanti, pinze e uno schermo televisivo. Che tipo di contatto emotivo è possibile stabilire con questo nuovo materiale se l'accesso fisico sembra così remoto?

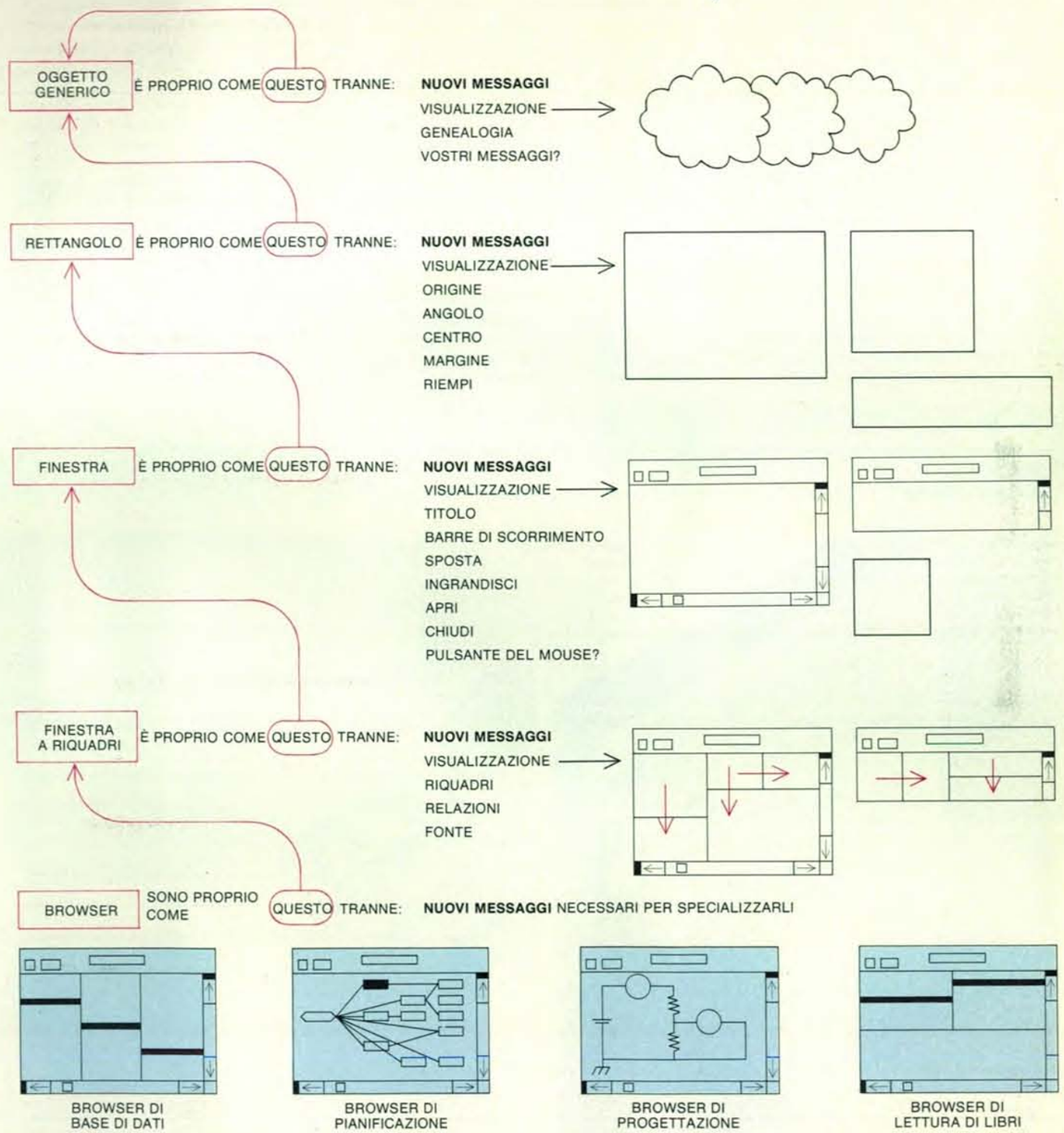
Uno sente la creta della computazione attraverso l'«interfaccia utente»: il software che fa da intermediario fra una persona e i programmi che fanno del calcolatore uno strumento per raggiungere un obiettivo specifico, sia che si tratti della progettazione di un ponte o della stesura di un articolo. Un tempo l'interfaccia utente era l'ultima parte del sistema a essere progettata. Oggi è la prima. Ci si è resi conto che essa è di primaria importanza perché, tanto per i principianti quanto per i professionisti, ciò che si presenta ai sensi di una persona *è* il calcolatore di quella persona. L'«illusione utente», così come i miei colleghi e io la battezzammo al Palo Alto Research Center della Xerox, è il mito semplificato che ognuno costruisce per cercare di spiegare le azioni del sistema e ciò che si dovrebbe fare poi.

Molti dei princìpi e dei dispositivi sviluppati per migliorare quell'illusione sono diventati ormai un luogo comune della progettazione del software. Forse il principio più importante è WYSIWYG (da *What you see is what you get,* ossia «quello che vedi è quello che ottieni»): l'immagine sullo schermo è sempre una fedele rappresentazione dell'illusione dell'utente. La manipolazione dell'immagine in un certo modo produce immediatamente qualcosa di prevedibile sullo stato della macchina (così come l'utente immagina quello stato). Una delle illusioni attualmente in voga ha «finestre», «menù», «icone» e un dispositivo di puntamento. I quadri di visualizzazione chiamati «finestre» permettono di presentare contemporaneamente sullo schermo numerose attività. Vengono visualizzati menù di possibili passi successivi; le icone rappresentano oggetti in forma di immagini concrete. Agendo sul dispositivo di puntamento (chiamato anche *mouse,* ossia «topo» per la sua forma) si sposta sullo schermo un puntatore, in modo da scegliere particolari finestre, voci di menù o icone.

Tutto questo ha dato luogo a una nuova generazione di software interattivo che trae vantaggio dall'illusione utente. L'obiettivo è quello di aumentare la capacità di simulazione dell'utente. Una persona esercita il massimo potere di leva quando la sua illusione può essere manipolata senza bisogno di ricorrere a intermediari astratti, come i programmi nascosti necessari per far funzionare anche un semplice elaboratore di testi. Quella che io chiamo leva *(leverage)* diretta viene esercitata quando l'illusione agisce come un *kit,* ossia come uno strumento con il quale risolvere un problema. La leva indiretta sarà acquisita quando l'illusione fungerà da «agente»: una estensione attiva del fine e degli obiettivi di una persona. In entrambi i casi il controllo del progettista del software su ciò che è essenzialmente un contesto teatrale è la chiave per creare un'illusione e accrescerne la percepita «amichevolezza».

I primissimi programmi di calcolatore furono progettati da matematici e da scienziati, i quali ritenevano che il loro compito dovesse essere lineare e logico. Il software risultò più difficile da plasmare di quanto essi avessero supposto. I calcolatori erano testardi: si ostinavano a fare quello che veniva loro detto e non quello che il programmatore intendeva dire. Di conseguenza una nuova classe di artigiani rilevò l'incombenza. Spesso questi piloti collaudatori del biplano binario non erano né matematici e neppure molto scientifici, ma erano profondamente coinvolti in un'avventura fantastica con quel materiale: quel genere di avventura che spesso porta a una nuova arte o a una nuova scienza. Il naturalista ha di fronte un'universo dato, e cerca di scoprirne le leggi. L'informatico promulga leggi in forma di programmi e il calcolatore dà vita a un nuovo universo.

Alcuni programmatori respirarono troppo profondamente l'inebriante atmosfera del crearsi un universo privato e diventarono quelli che l'insigne progettista Robert S. Barton chiamò «gli alti prelati di un basso culto». I più tuttavia si resero



La programmazione basata sull'eredità rivela il potere della descrizione differenziale. Un «oggetto» generico *(in alto)* viene visualizzato in forma di nube. Con l'oggetto indifferenziato si può fare un rettangolo dicendo, in pratica: «Voglio qualcosa proprio come questo, tranne...» e specificando poi certe proprietà quali la posizione del punto di origine (l'angolo sinistro in alto), la larghezza, l'altezza e così via. Un'ulteriore elaborazione dell'idea è una «finestra», un'area rettangolare dello schermo visualizzatore che dà una vista sull'uscita di un programma. Creando una finestra, è possibile farle «ereditare» proprietà pertinenti del rettangolo e aggiungere nuove strutture come barre di scorrimento (per far muovere la finestra sopra il materiale visionato) un titolo e dispositivi per variare dimensioni e posizione della finestra. Una finestra più complessa dotata di riquadri viene fatta aggiungendo nuovi metodi di visualizzazione per dare forma ai riquadri e stabilire comunicazioni fra di loro *(frecce in colore)*. Le finestre a riquadri si possono manovrare in modo da produrre dei *browser* (sfogliatori) ossia sistemi che permettono di recuperare risorse senza necessità di ricordarne i nomi. Qui sono rappresentati quattro esempi di browser *(in basso)*.

conto che altro è essere il dio di un universo e altro è dominarlo e cercarono fuori dal loro campo idee e ispirazione.

Un genere potente di programmi può fungere da ali o da catene. Le metafore più infide sono quelle che sembrano funzionare per un certo tempo: possono impedire, infatti, l'emergere di intuizioni ancora più potenti. Di conseguenza il progresso è lento - ma c'è. Si afferma un nuovo genere. Dopo qualche anno viene apportato un miglioramento significativo e, dopo qualche anno ancora, il miglioramento viene percepito non più soltanto come una «cosa vecchia migliore», ma come una «cosa quasi nuova» che porta direttamente al genere stabile successivo. È interessante che le cose vecchie e i relativi miglioramenti non scompaiono. Oggi prosperano validi rappresentanti di ogni era passata, come la programmazione in FORTRAN, un linguaggio vecchio di 30 anni e perfino l'antica «scrittura» nota come codice macchina diretto. Alcuni potrebbero considerare tali vestigia alla stregua di fossili viventi; altri farebbero notare che anche una specie antichissima potrebbe tuttora occupare una particolare nicchia ecologica.

Il campo dei calcolatori non ha ancora avuto il suo Galileo o il suo Newton, il suo Bach o il suo Beethoven, il suo Shakespeare o il suo Molière. Ciò che gli occorrerebbe innanzi tutto è un Guglielmo d'Occam, colui il quale disse: «*Entia non sunt multiplicanda sine necessitate*». L'idea secondo la quale vale la pena di fare tutto il possibile per eliminare la complessità in favore della semplicità aveva molto a che fare con la nascita di una scienza e di una matematica moderne, soprattutto dal punto di vista della creazione di una nuova estetica, un ingrediente vitale per qualsiasi settore in crescita. E quella che ci serve sia per giudicare il software di oggi sia per ispirare il software di domani è un'estetica basata sul «rasoio di Occam». Quanti concetti ci sono in effetti? E come si può sfruttare la metafora - magico processo che scopre analogie se non addirittura identità in strutture diverse - per ridurre la complessità?

Il matematico francese Jacques S. Hadamard scoprì, in una ricerca svolta su 100 matematici di primo piano, che questi per la maggior parte asserivano di non essersi serviti di simboli per esprimere il loro pensiero, ma che la loro impostazione concettuale si basava soprattutto su elementi visivi. Alcuni, compreso Einstein, risalivano, ancora più indietro, alla loro infanzia per richiamarsi a «sensazioni cinestetiche e muscolari». Le parti più antiche del cervello sanno cosa dire; quelle più nuove sanno come dirlo. Il mondo del simbolico può essere affrontato in maniera efficace solo quando l'accumulo di esempi concreti diventa tanto noioso da indurre a scambiarli con un'unica intuizione astratta.

In algebra il concetto di variabile, che permette di rappresentare e di trattare un'infinità di esempi come un'unica idea, costituì un progresso incredibile. Nel linguaggio la metafora accentua normalmente le analogie di cose completamente diverse come se fossero simili. Il rendersi conto che vari tipi di autocomparazione potevano essere ancora più validi fu un trionfo del pensiero matematico. Il calcolo differenziale di Newton e di Leibniz rappresenta idee complesse trovando modi per dire «Questa parte dell'idea è come quella parte, ove si eccettui...». I progettisti di sistemi di elaborazione al calcolatore hanno imparato a fare la stessa cosa con modelli differenziali, per esempio con metodi di programmazione che hanno la proprietà dell'eredità. Di recente sono stati concepiti modelli basati sull'idea della ricorsione, nei quali alcune delle parti sono veramente l'intero: è necessaria una descrizione dell'intero modello per generare la rappresentazione di una parte. Ne è un esempio la geometria frattale di Benoit B. Mandelbrot, in cui ogni sottoparte di una struttura è simile a ogni altra parte. Il caos è imprigionato nella legge.

Progettare le parti in modo che esse abbiano la stessa potenza dell'intero è una tecnica fondamentale del software di oggi. Una delle applicazioni più efficaci di tale tecnica è il progetto orientato all'oggetto. Il calcolatore viene diviso (concettualmente, sfruttandone la potenza di simulazione) in numerosi calcolatori, o oggetti, più piccoli, a ognuno dei quali può essere assegnato un ruolo, così come si fa con gli attori in un lavoro teatrale. Il passaggio alla progettazione orientata all'oggetto rappresenta un vero e proprio cambiamento di punto di vista - un cambiamento di paradigma - che porta con sé un enorme aumento della potenza espressiva. Un cambiamento analogo si ebbe quando le catene di molecole che galleggiavano sparse a caso in un oceano prebiologico videro aumentare di miliardi di volte la loro efficienza, la loro resistenza e le loro capacità energetiche allorché vennero racchiuse per la prima volta entro una membrana cellulare.

Le prime applicazioni di oggetti software furono tentate nel contesto della vecchia metafora dei linguaggi di programmazione sequenziali, e gli oggetti funzionavano come colonie di organismi unicellulari cooperanti. Le cellule sono, sì, una buona idea, ma accade sul serio qualcosa solo quando la cooperazione è abbastanza stretta da farle aggregare in supercellule: tessuti e organi. È possibile plasmare la struttura perpetuamente malleabile del materiale dei calcolatori in un «superoggetto»?

Il tabellone elettronico dinamico è un buon esempio di un superoggetto del genere, simile a un tessuto. È uno strumento di simulazione che fornisce una leva diretta di notevole potenza. Nel migliore dei casi i tabelloni elettronici combinano i generi affermatisi negli anni settanta (oggetti, finestre, redazione del tipo WYSIWYG e recupero finalizzato a un obiettivo *goal-seeking retrieval)* in una «cosa vecchia migliore» che ha molte probabilità di essere una delle «cose quasi nuove» per i progetti principali dei prossimi anni.

Un tabellone elettronico è un aggregato di oggetti simultaneamente attivi, organizzato di solito in una matrice rettangolare di celle simile alla carta fincata usata dai contabili. A ogni cella è assegnata una «regola», che specifica come stabilire il valore. Ogni volta che un valore viene cambiato in un punto qualsiasi del tabellone, tutti i valori che dipendono da esso vengono ricalcolati all'istante e vengono visualizzati i nuovi valori. Un tabellone elettronico è un universo tascabile simulato che conserva continuamente la propria struttura; è uno strumento valido per una gamma eccezionale di applicazioni. In questo caso l'illusione utente è semplice, diretta e potente. Si hanno poche sorprese mistificatorie, perché una cella può avere un valore solo quando le sia stata assegnata una regola.

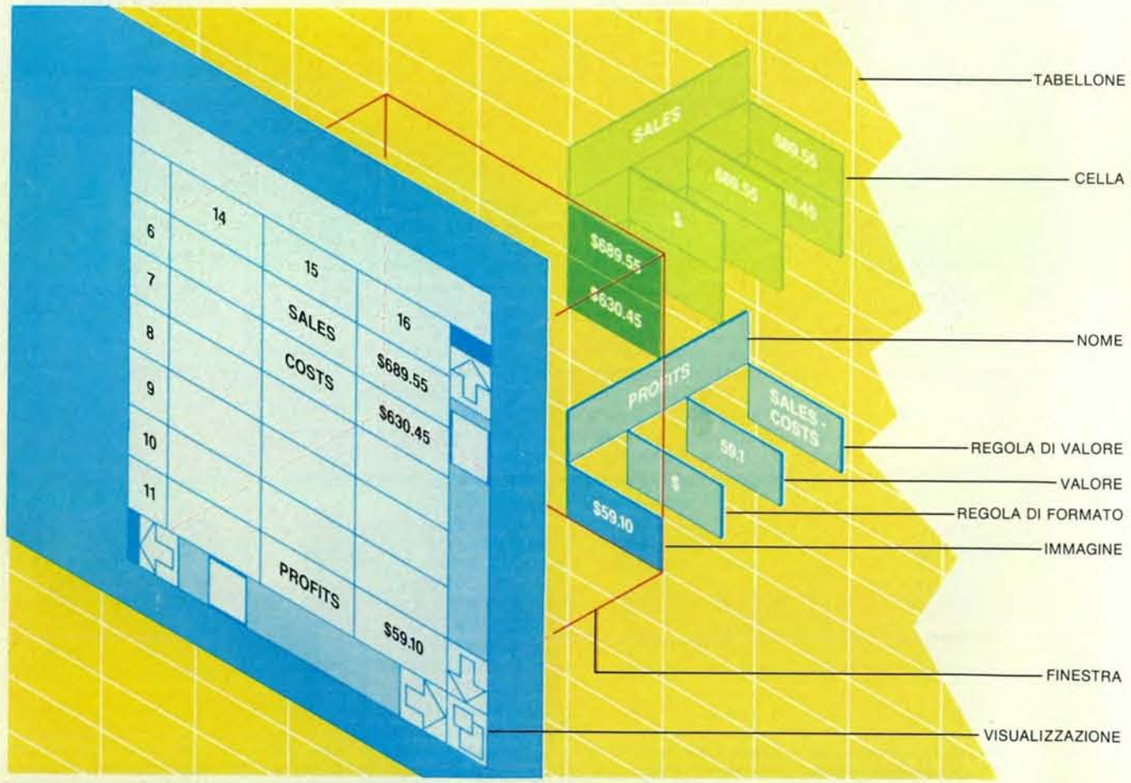
I tabelloni elettronici dinamici furono inventati da Daniel Bricklin e da Robert Frankston come reazione alla frustrazione provata da Bricklin quando si trovò costretto a lavorare con le vecchie versioni di carta fincata usata nelle scuole commerciali. I due furono sorpresi dal successo dell'idea e dal fatto che la maggior parte delle persone che acquistava il primo programma di tabellone elettronico (VisiCalc) se ne serviva per prevedere il futuro anziché per spiegare il passato. Cercando di sviluppare un «redattore intelligente», essi avevano creato uno strumento di simulazione.

Far sì che un tabellone elettronico esegua gli ordini di qualcuno è la cosa più semplice che possa esistere. La metafora visiva aumenta il riconoscimento di situazioni e strategie. La facile transizione dalla metafora visiva alla regola simbolica mette in azione tutta la potenza dei modelli astratti quasi senza preavviso. Una proprietà efficace è la capacità di rendere generica una soluzione «dipingendo» una regola contemporaneamente in molte decine di celle senza pretendere che gli utenti compiano delle generalizzazioni a partire dal loro livello concreto originale di pensiero.

Il tipo più semplice di regola fa della cella un oggetto statico come può essere un testo o molti testi. Una regola più complessa potrebbe essere una combinazione aritmetica di valori di altre celle, ricavati dalle loro posizioni relative o assolute o (molto meglio) dai nomi che sono stati ad esse assegnati. Una regola può verificare una condizione e stabilirne il valore a seconda del risultato. Versioni avanzate permettono al valore di una cella di essere recuperato mediante una ricerca euristica finalizzata all'obiettivo cosicché i problemi per i quali non esistono metodi diretti di soluzione possono ancora essere risolti con un processo di ricerca.

La verifica più valida di qualsiasi sistema non è data da quanto bene le sue caratteristiche si conformino a esigenze previste, ma da quanto bene si comporti quando si vuole fare qualcosa che il progettista non ha previsto. È una questione non tanto di possibilità quanto di evidenza: può l'utente vedere quello che si deve fare e procedere alla sua attuazione?

Si supponga di voler visualizzare dei dati con un insieme di colonne verticali la cui altezza sia normalizzata a quella del valore più elevato, e si supponga altresì che una tale struttura a istogramma non sia stata programmata nel sistema. Anche in un linguaggio di programmazione di alto livello ciò richiede un programma



Il tabellone elettronico dinamico è un *kit*, ossia uno strumento, di simulazione: si tratta cioè di un aggregato di oggetti di software, detti celle, che possono trarre valore l'uno dall'altro. La finestra sceglie per la visualizzazione una parte rettangolare del tabellone. Si può immaginare ogni cella come se avesse, dietro il tabellone, parecchi strati che ne calcolano il valore e che stabiliscono il formato della presentazione. Il nome della cella può essere battuto in una cella adiacente. Ogni cella ha una regola che può essere il valore stesso o un modo per calcolarlo; il valore può essere condizionato anche dallo stato delle celle in altre parti del tabellone elettronico. La regola di formato converte il valore in una forma adatta alla visualizzazione. L'immagine è il valore cui è stato dato un formato come viene visualizzato sul tabellone.

Skysensor Vixen

il primo telescopio intelligente

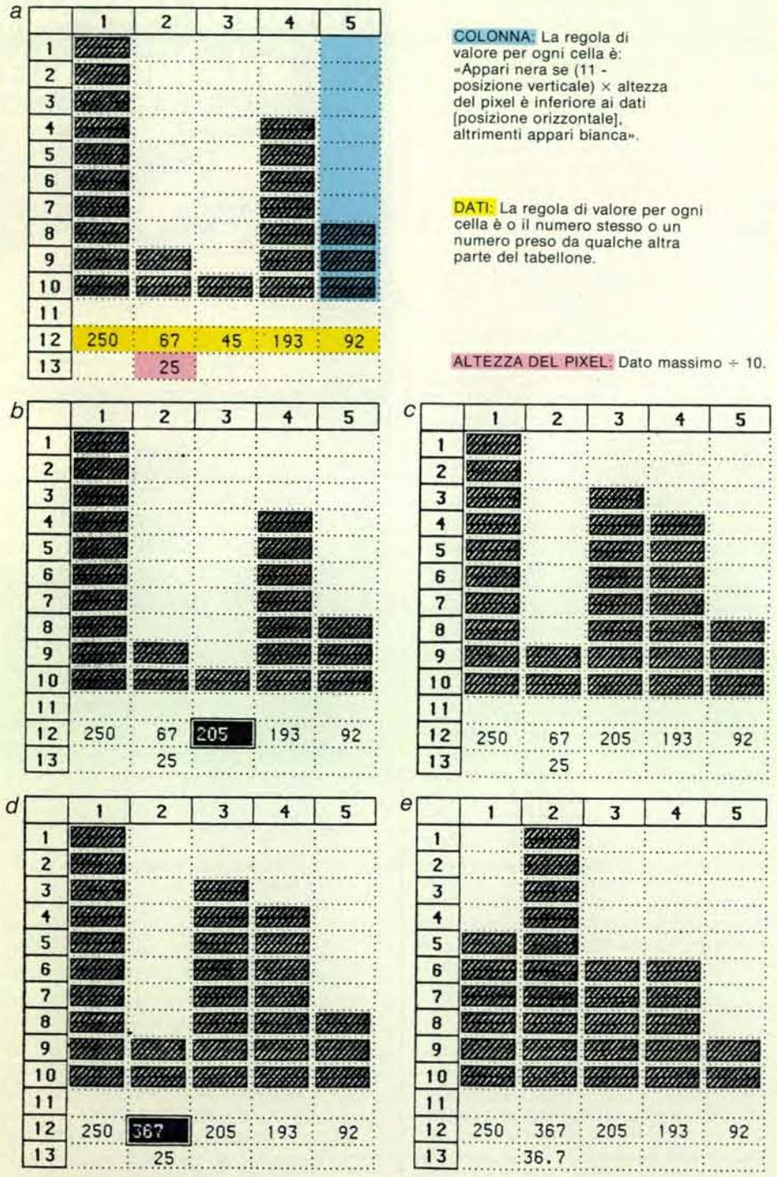
Finalmente l'osservazione del cielo è alla portata di tutti senza bisogno di approfondite conoscenze astronomiche e matematiche. Oggi Auriga presenta in Italia Skysensor Vixen, il microcomputer che pilota automaticamente i telescopi della serie Vixen Super Polaris. Volete osservare M31, la Galassia di Andromeda, distante oltre 2 milioni di anni luce? Impostate M31 sulla tastiera dello Skysensor ed il vostro telescopio Vixen, centinaia di volte più potente dell'occhio umano, si muoverà puntandosi rapidamente su questo stupendo oggetto celeste. E nella memoria dello Skysensor sono registrate le posizioni di 280 stelle e altre 450 galassie, nebulose, ammassi...
Alla vostra portata saranno anche il sole, la luna, i pianeti le comete e gli innumerevoli oggetti celesti che popolano il cielo notturno, oggetti che potrete osservare e fotografare con qualsiasi fotocamera reflex grazie alla vasta gamma di accessori Vixen.

Per ricevere documentazione e listini prezzi scrivere allegando L. 2.000 in francobolli a:

AURIGA

Auriga S. Via Zanella 56 20133 MILANO - Tel. (02) 7386045
Sala di esposizione a Roma:
Via Antonio Bertoloni 23/3 00197 ROMA - Tel.(06) 878243.

disordinato; in un tabellone elettronico la cosa è facile. Le celle fungono da «pixel» (elementi di immagine) del visualizzatore; una pila, o catasta, di celle costituisce una colonna. In una colonna che visualizza un terzo del valore massimo le celle del terzo inferiore sono nere e quelle dei due terzi superiori bianche. Ogni singola cella deve decidere se essere nera o bianca a seconda della propria posizione nella colonna: «Sarò nera se il punto in cui mi trovo nella colonna è inferiore ai dati che sto cercando di visualizzare; altrimenti sarò bianca *(si veda l'illustrazione di questa pagina)*.



È possibile costruire un istogramma con i materiali standard di un tabellone elettronico. Una colonna è fatta di celle, ognuna delle quali serve da *pixel*, o elemento di immagine. Una delle celle associate a ciascuna colonna contiene il dato, o valore, destinato a essere rappresentato dall'altezza della colonna corrispondente. Nell'ambito di una colonna tutte le celle sono governate dalla stessa regola. La grandezza rappresentata dall'altezza di un singolo pixel è il dato massimo diviso per il numero di pixel della colonna più alta; nell'istogramma *a* vi sono 10 pixel per colonna e ogni pixel rappresenta 25 unità. Ogni cella è nera se la sua posizione verticale nella colonna, moltiplicata per il numero di unità per pixel, è inferiore al dato relativo a quella colonna, altrimenti è bianca. Quando in una colonna si introduce un nuovo dato *(b)*, in quella stessa colonna appare una nuova serie di celle *(c)*. Se un dato nuovo è superiore al massimo precedente *(d)*, la serie di celle viene ricostruita *(e)* sulla base del nuovo numero di unità per pixel che in questo caso è 36,7.

Un altro esempio di tabellone elettronico è un sofisticato *browser* (sfogliatore) interattivo, un sistema progettato originariamente da Lawrence G. Tesler, allora al Palo Alto Research Center della Xerox. La ricerca con il *browser* è un modo piacevole di accedere a una base di dati organizzata gerarchicamente, indicando elenchi successivi. Il nome della base di dati viene battuto nel primo riquadro del visualizzatore, facendo sì che le aree per soggetto che ne costituiscono le immediate diramazioni siano recuperate e visualizzate nelle celle sotto il nome. È possibile scegliere una delle aree per soggetto indicandola con un *mouse;* l'area prescelta viene inserita in tal modo in testa alla colonna successiva, provocando a sua volta il recupero delle sue diramazioni. E si va avanti così finché non si raggiunge l'informazione desiderata *(si veda l'illustrazione nella pagina a fronte)*. È notevole il fatto che sia possibile programmare nel tabellone elettronico tutto il *browser* con sole tre regole.

L'intento di questi esempi non è quello di convincere tutti ad abbandonare ogni programmazione a favore dei tabelloni. I tabelloni elettronici attuali non sono ancora all'altezza della situazione, così come, forse, non lo è la metafora stessa del tabellone. Se però programmare significa scrivere passo passo delle ricette così come si è fatto da quarant'anni a questa parte, allora per la maggioranza delle persone si tratta di qualcosa che non è mai stato pertinente e che certo è obsoleto. I tabelloni elettronici, e soprattutto le loro estensioni del genere che ho suggerito, sono forti indizi che stili più validi per novizi ed esperti sono imminenti. Ciò significa forse che quella che potrebbe essere chiamata una «scuola guida» per saper leggere e scrivere con il calcolatore è tutto ciò di cui la maggior parte delle persone avrà mai bisogno - ossia che si dovrà apprendere soltanto a «guidare» i programmi applicativi e non ci sarà mai bisogno di imparare a programmare? Certo che no. Gli utenti devono essere in grado di adattare un sistema alle proprie necessità. Qualsiasi cosa in meno sarebbe altrettanto assurdo che pretendere di comporre dei saggi montando paragrafi scritti in precedenza.

Parlando di un mezzo proteiforme al massimo ho cercato di dimostrare quanto efficacemente il progetto conferisca potere di leva, specie quando il mezzo va modellato per essere uno strumento di leva diretta. È chiaro che, nel modellare attrezzature di software, i limiti del progetto sono quelli del creatore e dell'utente, non quelli del mezzo. La questione dei limiti del software acquista tuttavia importanza e centralità se si sostiene, come faccio io, che in futuro sarà disponibile un tipo più potente di leva indiretta, fornito da agenti personali: estensioni della volontà e degli scopi dell'utente ricavate dal materiale del calcolatore e in esso incorporate. Può il materiale dar vita alla mentalità? Pare che in un segno non vi sia nulla di simile alla mente. Come può una qualsiasi combinazione di segni, si tratti pure di segni dinamici e riflessivi, presentare una qualsiasi proprietà di mentalità?

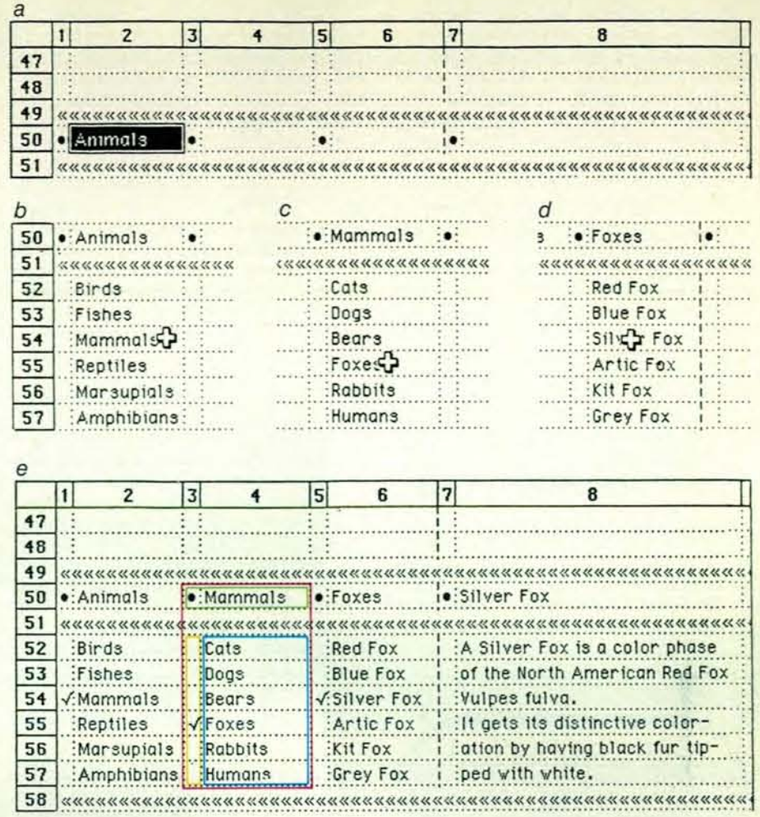
Anche gli atomi sembrano del tutto innocenti. Eppure la biologia dimostra che è possibile comporre materiali semplici in organizzazioni oltre modo complesse, capaci di autointerpretarsi e di automodificarsi dinamicamente. Sembra persino che alcune pensino! È difficile quindi negare certe possibilità di tipo mentale al materiale dei calcolatori, poiché il punto di forza del software è proprio (analogamente) la strutturazione cinetica di componenti semplici. I calcolatori «possono fare soltanto quello per cui sono programmati», ma lo stesso è vero anche per l'uovo fecondato impegnato a diventare un bambino. Tuttavia non si deve sottovalutare la difficoltà di scoprire un'architettura che generi mentalità. Lo studio della biologia era già in corso da alcune centinaia di anni quando, chiarite le proprietà del DNA e i meccanismi con cui si esprime, si vide che la cellula vivente è un'architettura in divenire. Inoltre, la biologia molecolare ha il vantaggio di studiare un sistema già montato e funzionante; per il compositore di software il calcolatore è come una bottiglia di atomi in attesa di essere modellati da un'architettura che egli deve inventare e poi imprimere dall'esterno.

Per proseguire l'analogia biologica, l'evoluzione può dire pochissimo sul mondo ai geni, e i geni possono dire ancora di meno al cervello in sviluppo. Tutti i livelli di competenza mentale sono presenti nell'oltre milione e mezzo di specie viventi. L'intervallo dei livelli va dal comportamento «cablato» in maniera così totale che l'apprendimento non è né necessario né possibile a modelli elaborati dall'esperienza e a una gamma di capacità così fluida da esigere un'organizzazione sociale stabile - una cultura - perché si realizzi il pieno potenziale adulto. (In altre parole, il modo adottato dai geni per far sì che un gatto prenda i topi è quello di programmare il gatto a giocare - e lasciare che i topi gli insegnino il resto.) Coloro i quali lavorano nel campo dell'intelligenza artificiale si accontentano in genere di tentare di imitare soltanto il primo tipo di comportamento, quello cablato. I risultati sono spesso chiamati sistemi esperti, ma in un certo senso sono solo i *jeans* «firmati» della scienza dei calcolatori. Non è che i loro inventori siano disonesti; pochi di essi rivendicano per il sistema più di quanto esso possa fare. Eppure l'etichetta «esperto» richiama alla mente una visione che poi delude quando si constata che i sistemi mancano di gran parte di quello che un comportamento esperto (o anche solo competente) è e come lo diventa.

Tre sviluppi hanno scarsissime probabilità di riuscita nel futuro immediato. Il primo è quello di poter costruire una mentalità adulta di tipo umano. Il secondo è quello di costruire la mentalità di un bambino e poi di «allevarla» in un ambiente capace di trasformarla in una mentalità adulta. Il terzo infine si basa sull'ipotesi che le tecniche attuali dell'intelligenza artificiale contengano i germi di un'architettura da cui costruire un tipo di mentalità autenticamente capace di diventare competente. Il fatto che le probabilità siano scarse non significa che l'impresa sia impossibile. Non è da escludere che il primo a realizzarsi sia il terzo sviluppo. Ancor prima che esso si realizzi ci saranno sistemi che appariranno in certo qual modo intelligenti e si comporteranno come tali, e alcuni di essi saranno effettivamente utili.

Come saranno gli agenti nei prossimi anni? L'idea di agente nacque con John McCarthy alla metà degli anni cinquanta, e il termine fu coniato da Oliver G. Selfridge qualche anno dopo, quando entrambi erano al Massachusetts Institute of Technology. Essi avevano in mente un sistema che, fissato un ben determinato obiettivo, eseguisse nei minimi particolari le giuste operazioni di calcolatore e potesse chiedere e ricevere consigli, offerti in termini umani, quando si fosse trovato in imbarazzo. Un agente sarebbe un *soft robot* che vive e opera nel mondo del calcolatore.

Che cosa potrebbe fare un tale agente? Centinaia di sistemi di recupero dei dati sono attualmente disponibili attraverso reti di calcolatori. Conoscere le arcane procedure di accesso di ogni sistema è quasi impossibile. Una volta ottenuto l'accesso, la tecnica del *browsing* (sfogliare) non può trattare più di circa 5000 voci.



IL RIQUADRO del browser ha una CELLA DI TESTA, CELLE DI BLOCCO e CELLE DI ELENCO.

LE CELLE DI BLOCCO notano il puntatore del *mouse* nella cella di elenco vicina e bloccano mostrando un segno di controllo.

LE CELLE DI ELENCO guardano il nome nella cella di testa e ricuperano le corrispondenti sottocategorie.

LA CELLA DI TESTA (tranne la prima) cerca la cella di blocco nell'elenco precedente e la copia.

Il *browser* permette l'accesso a una base di dati organizzata in maniera gerarchica, indicando semplicemente le voci di elenchi successivi. Per notizie sulla volpe argentata *(silver fox)*, per esempio, si batte nel primo riquadro *(a)* la parola «animali» *(animals)*. Nel riquadro appaiono le aree per soggetto della base di dati «animali» *(b)*. La selezione di «mammiferi» *(mammals)* fa apparire nel riquadro successivo quella sottocategoria *(c)*; la selezione di «volpi» *(foxes)* produce un elenco di volpi *(d)* e alla fine si ricupera la descrizione della volpe argentata *(e)*.

Ci vuole un agente che funga da bibliotecario solo per far fronte al semplice ordine di grandezza delle scelte. Esso potrebbe servire come una sorta di pilota, che si fa strada da una base di dati all'altra. Meglio ancora sarebbe un agente capace di presentare all'utente tutti i sistemi come un unico grande sistema, ma questo è un problema di notevole difficoltà. Un tenace «ricercatore» che per 24 ore al giorno cercasse le cose che sa che interessano l'utente e gliele presentasse come una pubblicazione personalizzata sarebbe più che benvenuto.

Gli agenti sono quasi inevitabilmente antropomorfici, ma non saranno mai umani e neppure, ancora per qualche tempo, molto competenti. Essi violano molti dei princìpi che definiscono una buona interfaccia utente e in particolar modo l'idea di conservare l'illusione utente. Sicuramente gli utenti saranno delusi se l'illusione prevista è quella di intelligenza, ma la realtà è sempre molto inferiore alle aspettative. Questa è la ragione principale del fallimento al quale sinora sono andati incontro i dialoghi condotti nella lingua di tutti i giorni, tranne quando il contesto del dialogo è rigorosamente limitato in modo da ridurre l'ambiguità.

Il contesto, ovviamente, è la chiave della situazione. L'illusione utente è teatro; è l'ultimo specchio. È il pubblico (l'utente) che è intelligente e che può essere indirizzato verso un particolare contesto. L'essenza del progetto dell'interfaccia utente è dare al pubblico gli indizi appropriati. Finestre, menù, tabelloni e così via forniscono un contesto che permette all'intelligenza dell'utente di continuare a scegliere la mossa successiva più appropriata. Un sistema basato su un agente dovrà fare la stessa cosa, ma la creazione di un'interfaccia che abbia qualche rassomiglianza con la mentalità umana richiederà una impostazione notevolmente più sofisticata.

Qualsiasi mezzo abbastanza potente da accrescere le possibilità dell'uomo è anche abbastanza potente da rovesciare il suo mondo. Fare in modo che la magia del mezzo operi a favore dei fini di qualcuno e non contro significa raggiungere l'alfabetizzazione. Nella sua accezione più semplice, alfabetizzazione significa scorrevolezza. La dimestichezza (conoscere la «grammatica») non è sufficiente. Le persone in grado di riconoscere un libro e le sue parole, una macchina per scrivere e la sua tastiera o un calcolatore e i suoi dispositivi d'ingresso e di uscita non sono «alfabete» a meno che non possano passare la maggior parte del tempo a trattare con il contenuto e non con la meccanica della forma.

Il calcolatore è un'automobile da guidare o un saggio da scrivere? La confusione è dovuta per lo più al fatto che si cerca di risolvere la questione a questo livello. La natura proteiforme del calcolatore è tale che esso può agire come una macchina o come un linguaggio da modellare e da sfruttare. È un mezzo che può simulare dinamicamente i particolari di qualsiasi altro mezzo, compresi i mezzi che non possono esistere materialmente. Non è uno strumento, anche se può agire come molti strumenti. È il primo «metamezzo», e come tale ha gradi di libertà, per quanto riguarda la rappresentazione e l'espressione, mai riscontrati in precedenza e finora poco studiati. Cosa ancor più importante, è divertente e quindi intrinsecamente degno di essere fatto.

Se i calcolatori possono essere automobili, allora l'alfabetizzazione relativa ai calcolatori a livello di corsi di scuola guida è certamente auspicabile. A dire il vero è in corso attualmente il tentativo di progettare interfacce utente che diano accesso alla potenza del calcolatore per mezzo di interazioni ancora più facili da imparare che non la guida di un'automobile. Programmi integrati per l'elaborazione di testi, per la grafica, per la simulazione, per il recupero delle informazioni e per la comunicazione interpersonale saranno la carta e la matita del prossimo futuro. L'alfabetizzazione a livello di carta e matita viene insegnata però all'asilo e alle elementari, e ciò significa che quella che può essere definita alfabetizzazione capace di creare segni nei calcolatori dovrebbe essere acquisita il più presto possibile; non si dovrebbe aspettare e farla imparare ai ragazzi nei soli sei mesi precedenti il diploma di scuola media superiore, come suggeriscono rapporti redatti di recente negli Stati Uniti da commissioni per l'istruzione. I ragazzi hanno bisogno di scarpe, biciclette, automobili e aeroplani «informatici» sin dal momento in cui incominciano a esplorare l'universo del sapere.

L'alfabetizzazione a livello di carta e matita inoltre non si ferma quando i ragazzi sanno tenere in mano una matita in modo da fare certi tipi di segni sulla carta. Una delle ragioni per le quali si insegna a leggere e a scrivere è certamente il fatto che le persone hanno bisogno di queste capacità per non perdersi nei meandri della vita quotidiana di questo nostro secolo. Ma vi sono anche obiettivi più elevati e più cruciali. Leggendo speriamo non solo di assimilare i fatti della nostra civiltà e di quelle che ci hanno preceduto, ma anche di imbatterci nella struttura e nello stile stessi del pensiero e dell'immaginazione. Lo scrivere ci fa scendere dagli spalti e ci fa entrare nel terreno di gioco; il sapere vecchio e nuovo diventa veramente nostro quando lo modelliamo direttamente.

In breve, noi ci comportiamo come se imparare a leggere e a scrivere aiutasse la gente a pensare in modo migliore e diverso. Supponiamo che partire con una bagaglio di conoscenze altresì vecchio di secoli sia più efficace di quanto lo sia partire da zero e costituisca un trampolino di lancio per idee nuove. Assumiamo che esprimere e modellare idee per mezzo della metafora e di altre forme retoriche renda le idee più completamente nostre e accresca la nostra capacità di imparare a nostra volta da altri. (Oliver Wendell Holmes disse: «Una volta ampliatasi alle dimensioni di idee più grandi, la mente non torna più alla sua dimensione originale».) Riteniamo che tutto questo sia importante, anche se leggere e scrivere sembrano molto difficili e richiedono anni per acquisirne una completa padronanza. La nostra società dichiara che questo tipo di alfabetizzazione non è un privilegio ma un diritto, non un'opzione ma un dovere.

Che cos'è allora l'alfabetizzazione relativa al calcolatore? Non è imparare a usare un elaboratore di testi, un tabellone elettronico o una moderna interfaccia utente: queste sono capacità a livello di carta e matita. L'alfabetizzazione relativa al calcolatore non è nemmeno imparare a programmare, cosa che si può fare in qualsiasi momento e in modi non più incoraggianti di quanto lo sia imparare la grammatica invece di imparare a scrivere.

L'alfabetizzazione relativa al calcolatore è un contatto con l'attività informatica abbastanza profondo da rendere fluente e piacevole l'equivalente informatico del leggere e dello scrivere. Come in tutte le arti, deve esserci con il materiale un rapporto d'amore. Se consideriamo l'apprendimento delle arti e delle lettere per tutta una vita come un punto di partenza per lo sviluppo dell'individuo e della società, non è forse opportuno fare ogni sforzo possibile per far partecipe l'informatica della nostra vita?

INFORMATICA

LE SCIENZE *edizione italiana di* **SCIENTIFIC AMERICAN**
ha pubblicato numerosi articoli su argomenti connessi al tema di questo fascicolo, fra cui:

LA CRITTOGRAFIA A CHIAVE PUBBLICA
di M. E. Hellman (n. 136)

BACKGAMMON AL CALCOLATORE
di H. Berliner (n. 144)

ALGEBRA AL CALCOLATORE
di R. Pavelle, M. Rothstein e J. Fitch (n. 162)

INTELLIGENZA ARTIFICIALE
di D. L. Waltz (n. 172)

LA MICROPROGRAMMAZIONE
di D. A. Patterson (n. 177)

STATISTICA AL CALCOLATORE
di P. Diaconis e B. Efron (n. 179)

TIPOGRAFIA DIGITALE
di C. Bigelow e D. Day (n. 182)

ELABORAZIONE DI TESTI IN PIÙ LINGUE
di J. D. Becker (n. 193)

Software per i sistemi intelligenti

La chiave per una risoluzione intelligente dei problemi consiste nel limitare la ricerca casuale delle soluzioni: i programmi di calcolo intelligente devono sfruttare le stesse «fonti di potenza» umane

di Douglas B. Lenat

Nel Nebraska, dirigendosi verso San Francisco, si arriva a un incrocio. È una sera d'estate. La strada continua diritta in avanti; a sinistra, una traversa si snoda attraverso i campi di grano e lo stesso fa verso destra, come si può vedere proteggendosi gli occhi dal sole. Non avendo nessuna carta stradale, si decide di prendere a caso una delle tre strade e, ad esempio, si gira a sinistra. Ben presto ci si trova a un altro incrocio, e poi a un altro ancora, e si è obbligati a fare una serie di scelte a caso; a un certo punto si finisce in un vicolo cieco e si deve ritornare all'incrocio precedente per imboccare una strada diversa. Se si è longevi e anche molto fortunati, forse alla fine si riuscirà a raggiungere San Francisco, ma le probabilità a sfavore di questo evento sono dell'ordine di 10^{30} a uno. Visto, però, che si sa come gira il mondo, non si deve per forza scegliere il percorso a caso ma, al primo incrocio, si sceglie di prendere a destra.

La maggior parte dei problemi, molti dei quali ben più interessanti di questo, possono essere presentati nella stessa forma: come la ricerca di un percorso da un qualche stato iniziale a uno stato finale che si vuole raggiungere. I problemi più interessanti hanno anche in comune la caratteristica di essere troppo complessi per essere risolti con una ricerca casuale, perché il numero di scelte cresce in modo esponenziale a mano a mano che ci si allontana dal primo incrocio, o punto di decisione. Classico è l'esempio degli scacchi, per i quali è stato stimato a 10^{120} il numero di possibili posizioni della scacchiera. Un buon giocatore, però, riduce a proporzioni abbordabili il problema di scegliere la mossa successiva, considerando solo quel centinaio di posizioni che corrisponde alle più promettenti linee di attacco. È qui che sta, a mio giudizio, l'essenza dell'intelligenza: scoprire come risolvere problemi altrimenti inaffrontabili limitando la ricerca delle soluzioni.

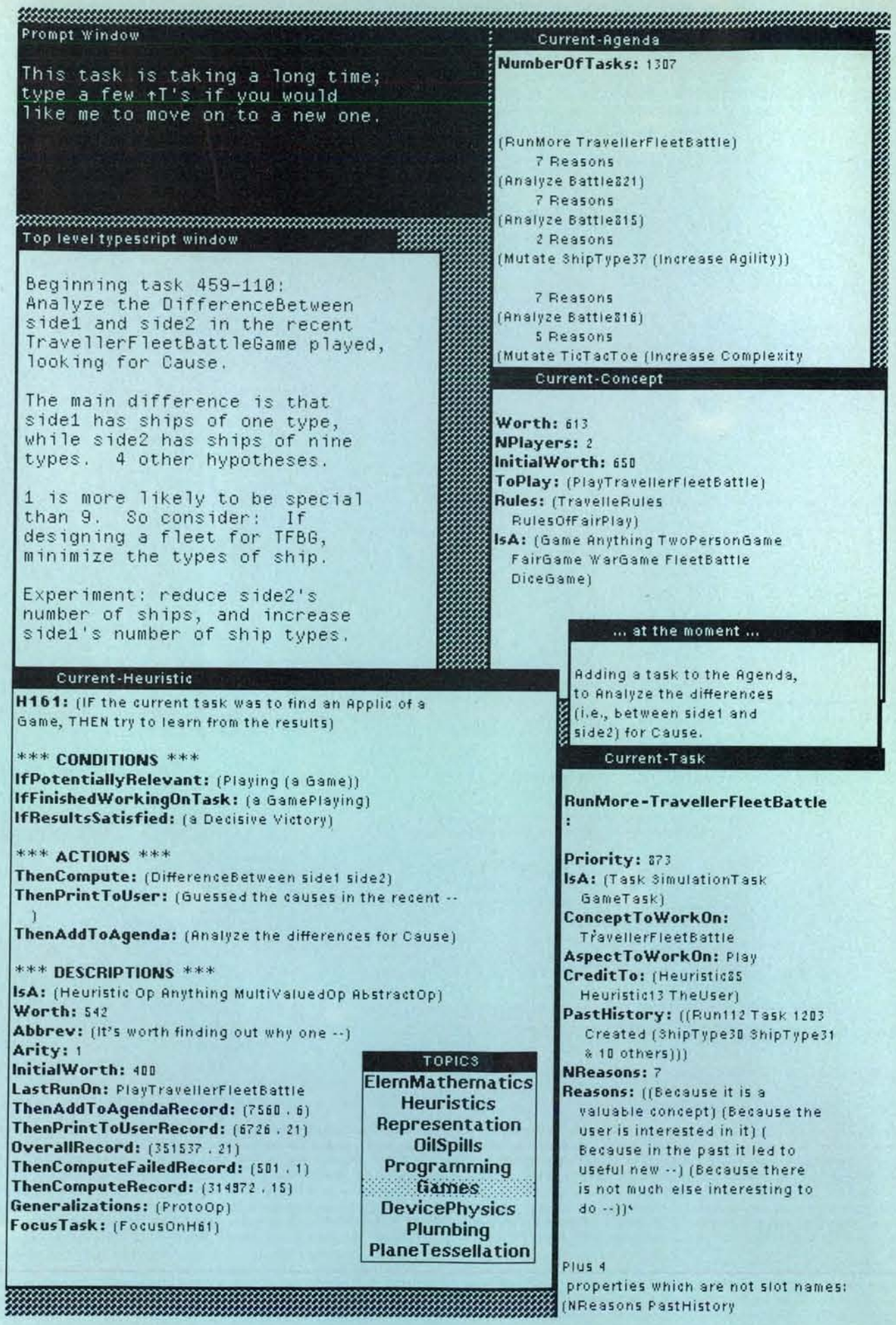
Per circa 30 anni un piccolo gruppo di ricercatori ha tentato, con più o meno successo, di programmare dei calcolatori per risolvere in modo intelligente dei problemi. Verso la metà degli anni settanta, dopo vent'anni di progressi di una lentezza scoraggiante, gli operatori nel nuovo campo dell'intelligenza artificiale erano arrivati a una conclusione fondamentale sul comportamento intelligente in genere: esso richiede un'enorme quantità di conoscenze, che è spesso scontata per la gente, ma che deve essere totalmente fornita a un calcolatore. All'incrocio nel Nebraska, un essere umano sa che San Francisco è a ovest, che la sera il sole si trova a ovest nel cielo e che, dirigendosi verso il sole, si va grosso modo nella direzione giusta. Non avrebbe quindi la necessità di esplorare gli altri due possibili percorsi.

La relativa semplicità di questo problema non è rappresentativa di altri compiti quotidiani che la gente porta a termine senza nemmeno starci a pensare. La comprensione di semplicissimi passi nella propria lingua madre, per esempio, richiede anch'essa una conoscenza del contesto, di chi parla e del mondo in genere, conoscenza che trascende di gran lunga la capacità degli attuali programmi per calcolatore. Il ruolo centrale che la conoscenza svolge nell'intelligenza spiega perché finora i programmi di maggior successo sono stati «sistemi esperti» che operano in campi altamente specializzati, come la diagnosi della meningite, e i programmi di gioco. Per contro, i primi tentativi di progettare un «risolutore generale di problemi» partivano dal presupposto che il cuore dell'intelligenza stia in una capacità di ragionamento applicabile in tutti i campi. Essi si sono dimostrati meno fruttuosi e sono stati ormai per lo più abbandonati.

La gente, quando affronta un problema complesso, punta su vari modi di usare la propria conoscenza della regolarità del mondo per limitare la ricerca di una soluzione, modi che io chiamo «fonti di potenza». Può invocare teoremi matematici o regole pratiche meno formali; può suddividere il problema in sottoproblemi più gestibili, oppure può ragionare per analogia con problemi che sono già stati risolti. I programmi per calcolatore, per quello che già possono esibire come intelligenza, si basano su alcune di queste stesse fonti di potenza e il futuro dell'intelligenza artificiale sta proprio nella possibilità di trovare modi per attingere a quelle che solo ora hanno cominciato a essere sfruttate.

Parecchi programmi scritti nei primi vent'anni di ricerca sull'intelligenza artificiale dipendevano molto da metodi formali di ragionamento. Quando un compito è ben definito in un campo assai ristretto, questi metodi possono dimostrarsi molto efficaci nel potare o addirittura eliminare l'albero di ricerca. Per

Un programma intelligente viene eseguito da una macchina che visualizza vari aspetti dell'operazione in finestre. Il programma EURISKO, scritto dall'autore e dai suoi collaboratori, è stato applicato a numerosi argomenti, compresi quelli riportati nella piccola finestra in fondo al video; qui progetta una flotta di navi per combattere nel Traveller T.C.S., un gioco di simulazione bellica. Nella base di conoscenza del programma figurano le complesse regole del gioco e regole euristiche generali per guidarlo nella ricerca di schemi sempre migliori. EURISKO ha appena simulato una battaglia nella quale il giocatore 1 («side1») ha decisamente sconfitto il giocatore 2 («side2») e la regola euristica in corso lo ha guidato ad apprendere dai risultati della battaglia, mediante un'analisi delle differenze tra le due flotte, la causa della vittoria del giocatore 1. La differenza principale è che il giocatore 1 ha solo un tipo di nave, ed EURISKO ipotizza che sia preferibile ridurre il numero di tipi di nave e suggerisce un esperimento per verificare l'ipotesi. Durante la competizione la flotta di EURISKO, costituita principalmente di piccole navi veloci, ha sconfitto le flotte dei giocatori umani.



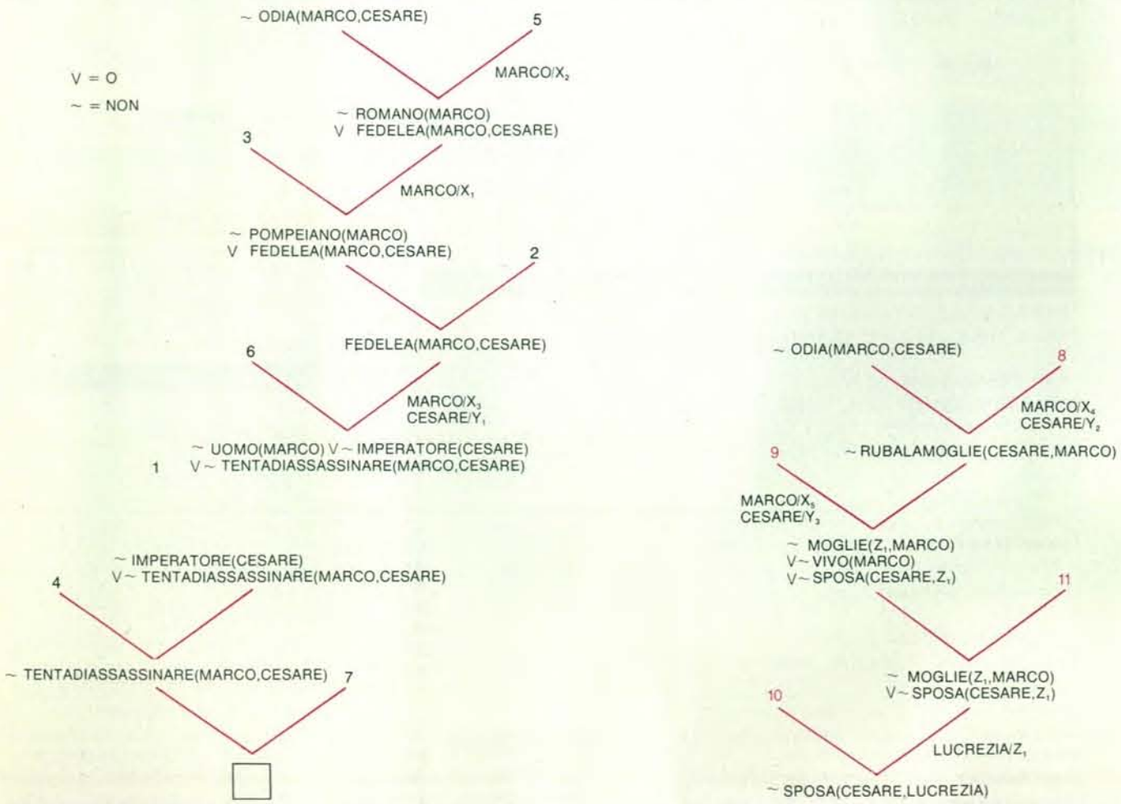
esempio, nessuno ha più bisogno di perder tempo a cercare un modo per trisecare un angolo o il metodo migliore per calcolare il moto di un proiettile, perché teoremi e algoritmi ormai acquisiti hanno risolto quei problemi una volta per tutte.

Uno dei metodi formali più popolari è stata la deduzione logica per mezzo di una tecnica, detta risoluzione, in cui le dimostrazioni sono condotte per assurdo. Per applicare la risoluzione si deve prima tradurre l'enunciato da dimostrare nel formalismo logico del calcolo dei predicati. L'enunciato viene poi negato e la negazione è «risolta» con una serie di assiomi, enunciati di cui è nota la verità per l'area di quel particolare problema. Se le inferenze tratte combinando la negazione con gli assiomi portano a una contraddizione, la negazione deve essere falsa e l'enunciato originale deve pertanto essere vero.

«MARCO ODIAVA CESARE?»

1 UOMO(MARCO)
2 POMPEIANO(MARCO)
3 ~ POMPEIANO(X_1) V ROMANO(X_1)
4 IMPERATORE(CESARE)
5 ~ ROMANO(X_2) V FEDELEA (X_2,CESARE) V ODIA(X_2,CESARE)
6 ~ UOMO(X_3) V ~ IMPERATORE(Y_1) V ~ TENTADIASSASSINARE (X_3,Y_1) V ~ FEDELEA(X_3,Y_1)
7 TENTADIASSASSINARE(MARCO,CESARE)
8 ~ RUBALAMOGLIE(Y_3,X_4) V ODIA(X_4,Y_2)
9 ~ MOGLIE(Z_1,X_5) V ~ VIVO(X_5) V ~ SPOSA(Y_3,Z_1) V RUBALAMOGLIE(Y_3,X_5)
10 MOGLIE(LUCREZIA,MARCO)
11 VIVO(MARCO)

1 Marco era un uomo
2 Marco era un pompeiano
3 Tutti i pompeiani erano romani
4 Cesare era un imperatore
5 Tutti i romani o erano fedeli a Cesare o lo odiavano
6 I sudditi tentano di assassinare solo gli imperatori ai quali non sono fedeli
7 Marco tentò di assassinare Cesare
8 Un uomo odia chi gli ruba la moglie
9 Se la moglie di un uomo che è vivo sposa un altro, allora questi ha rubato la moglie al primo
10 Lucrezia era la moglie di Marco
11 Marco era vivo



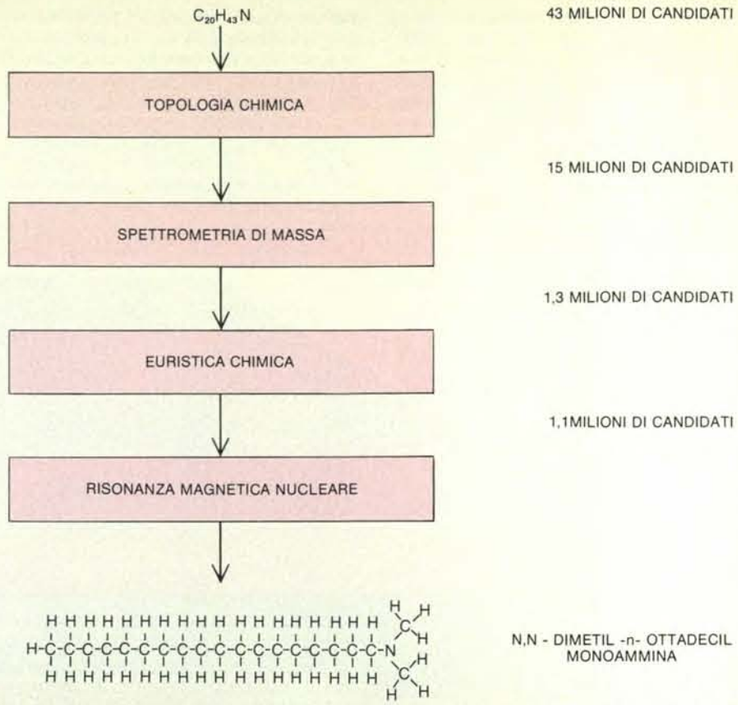
La risoluzione, una tecnica dimostrativa della logica formale, può servire a dedurre dalle domande le risposte, ma comporta un proibitivo dispendio di tempo quando si tratta di un problema complesso. La risoluzione è una dimostrazione condotta per assurdo; si dimostra che un'ipotesi è valida dimostrando che la sua negazione, quando è confrontata con assiomi, o enunciati che si sa essere veri, porta a contraddizione. Dapprima l'ipotesi negata e gli assiomi sono tradotti nella notazione logica: proposizioni complesse che hanno la forma di disgiunzione di proposizioni semplici. Si cerca poi nell'insieme degli assiomi un assioma che contenga una proposizione, la quale, per opportune sostituzioni delle variabili, contraddica uno dei disgiunti nell'ipotesi negata. Quando i due enunciati sono «risolti», i disgiunti in contraddizione si annullano. La procedura viene successivamente ripetuta con l'enunciato risultante; alla fine, se l'ipotesi originaria è valida, il processo termina con una contraddizione pura e semplice. Nell'esempio, l'ipotesi è «Marco odiava Cesare». Nel caso ideale in cui si abbia una scarsa conoscenza del mondo *(a sinistra)*, solo in un assioma *(5)* c'è una proposizione che contraddice la negazione dell'ipotesi e un programma di calcolatore può rapidamente completare una dimostrazione. Quando è disponibile una maggior quantità di informazione *(assiomi in colore)*, ivi compresa una diversa causa dell'odio *(8)*, il programma può scegliere l'assioma sbagliato e arrivare a un punto morto nel quale non si genera nessuna contraddizione *(a destra)*. In un problema del mondo reale, il numero di possibili scelte è funzione esponenziale del numero degli assiomi, che è molto grande, ed è impensabile trovare una soluzione cercando alla cieca. L'esempio è stato fornito da Elaine Rich.

Nel 1964, I. A. Robinson dimostrò che il metodo della risoluzione è «completo»: in ogni caso esso finirà col generare una contraddizione se il teorema originale è vero. (Se il teorema originale è falso, non è garantito che la serie di inferenze prodotta dalla risoluzione abbia termine.) Il lavoro di Robinson ha dato origine a un'impetuosa ondata di attività, che è durata per un decennio ed è stata centrata sull'applicazione del metodo della risoluzione e di altri metodi formali a esso strettamente collegati per la soluzione automatica di teoremi. Risulta che i programmi per calcolatore sono capaci di dimostrare enunciati di moderata difficoltà e che il metodo della risoluzione può essere adattato anche a programmi intesi a rispondere a domande piuttosto che a dimostrare teoremi. La grande pecca di questo metodo è che è soggetto a «esplosione combinatoria»: il numero di risoluzioni che il programma deve tentare cresce in modo esponenziale con la complessità del problema. Programmi che applicano con successo la risoluzione a piccoli casi di prova hanno costantemente fallito la «scalata» a più interessanti problemi del mondo reale.

La stessa difficoltà affligge il software basato su una differente tecnica logica, detta induzione strutturale. A questi programmi viene fornita una grande quantità di dati sugli oggetti di un particolare campo e viene loro detto di costruire un albero di decisione per discriminare tra oggetti. Il problema con gli algoritmi a induzione strutturale, però, è che essi non includono alcuna informazione che li renda in grado di decidere quali variabili sono importanti o di affrontare dati di particolare vistosità o casi eccezionali. Se, in un campo, il numero degli oggetti e delle caratteristiche a essi associate è grande, l'albero di decisione, generato dal programma, diventa troppo ingombrante.

Per poter usare il ragionamento formale come unica fonte di potenza in un programma, il problema deve essere piccolo. Un'applicazione di un metodo formale che si può rivelare fruttuosa nel prossimo futuro è la simulazione del ragionamento fisico qualitativo. John Seely Brown e Johan de Kleer dello Xerox Palo Alto Research Center hanno scritto un programma che, per mezzo di equazioni qualitative, costruisce un modello dei cambiamenti in una valvola per la regolazione della pressione. Se, ad esempio, si dice al programma che la pressione sul lato sinistro della valvola è aumentata, le equazioni vengono modificate di conseguenza e il programma prevede il cambiamento di pressione sull'altro lato della valvola e lo stato di equilibrio finale del sistema. Questo è un esempio molto semplice, ma si adotta un'impostazione analoga nell'analisi e nella progettazione di circuiti elettrici.

La maggior parte dei problemi più interessanti, però, non si può risolvere affidandosi solo al ragionamento formale. La forza dei metodi logici sta nel fatto che essi rappresentano il mondo con simboli che si possono manipolare in modi conosciuti a fondo (come la risoluzione) per produrre inferenze. Questa forza è però anche la loro maggior debolezza: molti tipi di conoscenza, tra cui la conoscenza incerta e incompleta che caratterizza la maggior parte dei problemi del mondo reale, non si prestano a essere rappresentati attraverso precisi formalismi logici. I programmi che si fondano esclusivamente sulla logica catturano solo un parte dell'attività mentale mobilitata da una persona intelligente nel tentativo di risolvere un difficile problema.



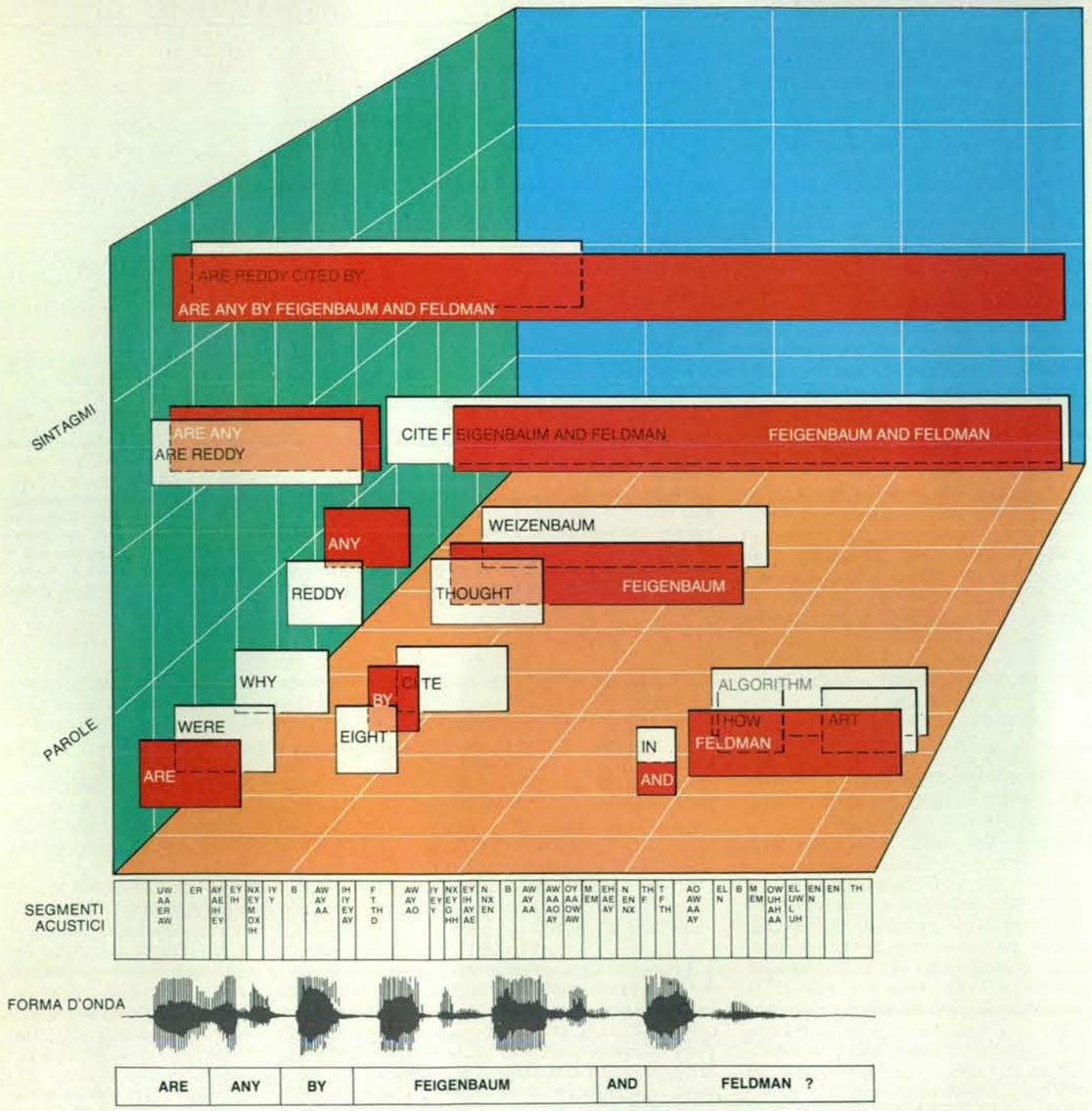
I sistemi esperti sfruttano la conoscenza di un campo specializzato per restringere la ricerca di soluzioni. Il primo sistema esperto e uno dei più riusciti, DENDRAL, si basa sull'interazione sinergica di quattro tipi di conoscenza per restringere il campo di strutture candidate per una particolare molecola organica. Se si conosce solo la formula molecolare $C_{20}H_{43}N$, sono possibili 43 milioni di configurazioni degli atomi. Una conoscenza della topologia chimica di base, ad esempio del fatto che un atomo di carbonio ha quattro legami, riduce il numero di candidati a 15 milioni. Il tipo di frammentazione di una molecola in uno spettrometro di massa, con la conoscenza euristica delle strutture più stabili e perciò più plausibili, riduce ancora la ricerca. Infine, i dati di risonanza magnetica nucleare permettono di identificare la struttura giusta.

Decine di vasti programmi sono attualmente applicati a difficili problemi tecnici nei campi più disparati, come la diagnosi medica, la programmazione di esperimenti genetici, la prospezione geologica e la progettazione meccanica. La fonte primaria di potenza, in questi sistemi esperti, è il ragionamento informale basato su una vasta conoscenza accuratamente selezionata da esperti umani. Nella maggior parte dei programmi la conoscenza è codificata sotto forma di centinaia di regole pratiche se-allora, o euristiche. Tali regole restringono il campo della ricerca, guidando l'attenzione del programma verso le soluzioni più probabili. Inoltre - e questo distingue i programmi a guida euristica da quelli che si affidano a metodi più formali - i sistemi esperti sono in grado di spiegare tutte le loro inferenze in termini accettabili da un essere umano. La spiegazione può essere data dal fatto che le decisioni si fondano su regole insegnate da esperti umani piuttosto che sulle regole astratte della logica formale.

Prendiamo in considerazione MYCIN, un programma sviluppato da Edward H. Shortliffe della Stanford University per diagnosticare le infezioni batteriche del sangue. Il problema consiste nello stabilire quale di molti possibili organismi potrebbe essere responsabile di una particolare infezione e nel raccomandare una certa terapia sulla base della diagnosi fatta. Per raggiungere questo risultato, MYCIN sfrutta una base di conoscenza di 500 regole euristiche, di cui questo è un esempio tipico: «Se (1) l'organismo è gram-positivo e (2) la sua morfologia è quella di un cocco e (3) la forma dell'accrescimento è a blocchi, allora ci sono elementi per sostenere (0,7) che l'identità dell'organismo è quella di uno stafilococco». Durante il funzionamento, il programma conversa con l'utente, chiedendo

ulteriori informazioni sul paziente che gli consentiranno di applicare regole differenti, e a volte suggerendo prove di laboratorio. In qualsiasi momento l'utente può chiedere a MYCIN di giustificare una domanda o un'inferenza, facendo riferimento alla regola a cui si è appellato. Il programma si è dimostrato capace di prestazioni pari a quelle di esperti umani.

I sistemi esperti attingono, oltre che al ragionamento euristico, ad altre fonti di potenza, alcune delle quali fanno talmente parte del senso comune che raramente la gente pensa a esse in modo cosciente. Molti programmi, ad esempio, sono in grado di focalizzare la loro ricerca in virtù dell'orientamento che hanno verso obiettivi più o meno specifici. Il programma MYCIN è ancora una volta esemplare: sulla scorta di un'iniziale informazione generica sul paziente, esso ragiona in senso inverso partendo dall'obiettivo di trovare l'identità dell'organismo che genera la malattia e ponendo domande per scoprire i sintomi specifici che potrebbero sostanziare una diagnosi. Avendo stabilito, ad esempio, che «l'organismo è gram-positivo», MYCIN indagherebbe, senza ulteriore ricerca, sulla morfologia dell'organismo infettante per decidere se potrebbe essere uno stafilococco. Il fatto di essere indirizzato a un obiettivo non significa che un programma abbia una sequenza decisionale «vincolata», come suggeriscono quanti sostengono che i sistemi esperti non dimostrano in realtà intelligenza; programmi come MYCIN sono realmente adattabili a situazioni non previste dal programmatore, che non determina preliminarmente in modo rigoroso l'uso che un programma farà delle sue conoscenze.



La «lavagna» permette di organizzare una grande quantità di conoscenze in un programma intelligente. L'informazione viene immagazzinata in moduli indipendenti, ciascuno dei quali controlla solo una piccola regione della lavagna e viene attivato solo quando un altro modulo invia in quella regione delle informazioni. La struttura modulare aiuta a risolvere il problema di decidere quale parte della conoscenza base debba essere applicata in un dato momento. Nell'esempio che è stato adattato da un sistema di comprensione del parlato elaborato da Raj Reddy, Lee D. Erman e collaboratori, l'asse orizzontale rappresenta il tempo, a partire dall'inizio di un'emissione vocale, e l'asse verticale rappresenta il livello di astrazione, a partire da onde sonore per arrivare a un enunciato completo. La terza dimensione indica il livello di certezza associato a ciascuna ipotesi posta sulla lavagna; le più plausibili tra le molte ipotesi possibili ogni volta e a ogni livello di astrazione sono in primo piano. La lavagna permette l'interazione tra moduli di conoscenza di livelli diversi; per esempio, quando il programma inferisce dall'intonazione che la frase è interrogativa, tale informazione guida la formazione di ipotesi relative ai livelli della parola o del sintagma.

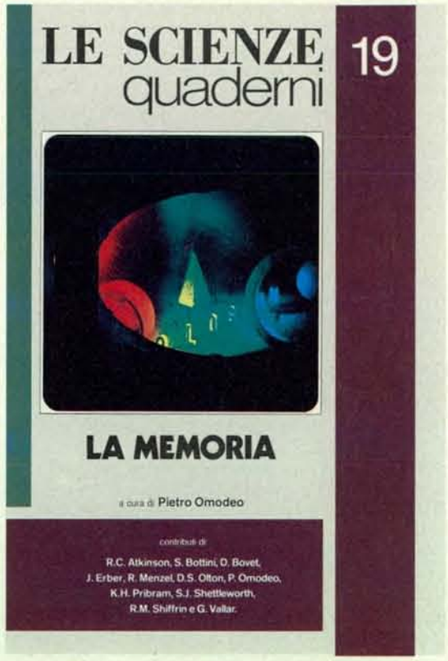
Un'altra potente strategia sfruttata dagli esseri umani intelligenti, inclusi i progettisti di software, è quella di smembrare un problema complesso in sottoproblemi più facilmente affrontabili: è la strategia del *divide et impera*. Un gruppo della Carnegie-Mellon University ha costruito quattro programmi collegati, ciascuno dei quali è guidato dall'euristica a riscoprire ben note leggi della fisica e della chimica. Questi programmi mettono il gruppo in grado di compiere progressi verso la comprensione e la meccanizzazione di diversi aspetti del processo di formazione delle teorie scientifiche; alla fine i ricercatori salderanno queste soluzioni in un unico modello dell'intero processo.

In un senso relativamente stretto, l'impostazione del *divide et impera* è implicita nello stesso software intelligente. I programmi diretti dall'obiettivo suddividono la ricerca in sotto-obiettivi più o meno indipendenti (nodi dell'albero di ricerca). A un livello superiore, sistemi a guida euristica distinguono il problema stesso dal metaproblema, che è la difficoltà di decidere in qualsiasi ben determinato momento quale delle centinaia di differenti regole andrebbe «attivata». Il metaproblema è risolto separatamente da un processo spesso complicato, che a volte richiede una propria euristica, di accoppiamento tra stato della ricerca e precondizioni; queste sono la parte «se» di una regola se-allora.

Anche i metodi formali, pur non essendo il motore dell'inferenza, possono essere utili nella gestione di un sistema esperto. Per esempio, alcuni sistemi si affidano a procedure logiche o statistiche nel decidere quando non è più conveniente continuare una ricerca. Inoltre, dato che l'euristica se-allora in un sistema esperto normalmente non esprime relazioni che siano sempre vere, ogni regola può avere associata una valutazione di affidabilità («0,7» nel precedente esempio con MYCIN). Le valutazioni collegate a ogni stadio di una sequenza di inferenze vengono combinate e producono una misura di affidabilità per la conclusione finale. Questo avviene usando la legge di Bayes o qualche altra procedura formale di teoria della probabilità.

Tutte le regole in un sistema esperto possono essere semplici e a volte, tra di esse, l'organizzazione è scarsa o addirittura nulla. Eppure l'insieme nel suo complesso è capace di portare a compimento difficili compiti tecnici con il livello di competenza di un esperto. Si tratta di una forma di sinergia: l'intero è maggiore della somma delle parti. La sinergia è così

A dicembre sarà disponibile in edicola e in libreria LA MEMORIA un nuovo quaderno di «Le Scienze» indispensabile per fare il punto su un argomento che per decenni ha messo alla prova neurofisiologi e biologi e il cui studio ha ricevuto nuovo impulso dai modelli teorici elaborati nell'ambito delle attuali tecnologie fisiche e informatiche. Il quaderno è arricchito da numerosi contributi originali.



Otto QUADERNI all'anno, ogni mese da ottobre a maggio. Formato cm 21 × 29. Prezzo di copertina: L. 4500

Il controllo della memoria a breve termine di R. C. Atkinson e R. M. Shiffrin
La neuropsicologia della memoria a breve termine di G. Vallar
La memoria nei piccoli roditori di D. Bovet
La memoria spaziale di D. S. Olton
Apprendimento e memoria nelle api di R. Menzel e J. Erber
Come gli uccelli ritrovano le provviste nascoste di S. J. Shettleworth

inoltre scritti appositamente per questo quaderno:
Come ricordano le macchine e gli animali di P. Omodeo
L'informazione olografica di R. Nobili
Un modello di memoria associativa di S. Bottini
La sperimentazione sulla memoria e sull'apprendimento di P. Omodeo

Il quaderno in edicola questo mese è:
UOMINI E NUMERI a cura di Ettore Picutti

estesa da essere data per scontata, ma quasi tutti i sistemi esperti si affidano a essa come fonte di potenza.

Uno dei programmi intelligenti di maggior successo - DENDRAL, scritto da Edward A. Feigenbaum e collaboratori a Stanford verso la fine degli anni sessanta - fu anche il primo sistema esperto sviluppato. Insieme al suo successore, GENOA, è attualmente usato nei laboratori di chimica organica di tutto il mondo ed è in grado di dedurre la struttura di molecole organiche dagli spettri di massa, dai dati di risonanza magnetica nucleare e da altri tipi di informazione.

Come MYCIN, DENDRAL è essenzialmente diagnostico. Un tipo completamente diverso di sistema esperto è quello che cerca di scoprire nuove informazioni o di riscoprire informazioni già note da principi base. Un esempio di programma del genere è EURISKO, che ho sviluppato a Stanford insieme ai miei studenti. Dopo avergli fornito una quantità relativamente limitata di informazioni base, lo abbiamo lasciato spaziare in campi tra loro diversi come la teoria degli insiemi, un gioco di simulazione bellica, la programmazione di un calcolatore e l'eliminazione delle scorie chimiche.

EURISKO è guidato nella sua ricerca, fatta di sintesi, di analisi e di valutazione di concetti nuovi, da centinaia di regole euristiche piuttosto generiche. Una di queste è «osserva i casi estremi», regola che portò EURISKO, mentre stava meditando sulla funzione «divisori di» nella teoria degli insiemi, a considerare i numeri che hanno solo pochi divisori. Così facendo, EURISKO riscoprì i numeri primi, che sono numeri con solo due divisori, e il fatto che i fattori di qualsiasi numero rientrano in un unico insieme di primi. La stessa semplice regola si rivelò inestimabile quando EURISKO e io ci impegnammo nel gioco di simulazione bellica Traveller T.C.S., il cui obiettivo è di progettare una flotta che sconfigga gli avversari in battaglie ingaggiate seguendo un grande numero di regole rigide. Dopo aver preso in considerazione le regole, EURISKO produsse una flotta fatta quasi interamente d'unità d'attacco piccole e rapide, abbastanza simili a delle motosiluranti, tra cui una nave così veloce e così minuscola da risultare praticamente indistruttibile. Le persone che partecipavano al gioco si presero beffe di questa strategia e misero in campo flotte convenzionali, con un maggior equilibrio per quanto riguarda le dimensioni delle navi: EURISKO vinse.

Un'altra regola euristica di ampia applicabilità è «fai coincidere», che porta il programma a prendere in esame che cosa avviene a una funzione di due variabili x e y quando a queste variabili è assegnato lo stesso valore. Dopo che EURISKO aveva già derivato le funzioni di addizione e di moltiplicazione dalla teoria degli insiemi, la regola «fai coincidere» lo spinse a scoprire il raddoppio (x più x) e il quadrato (x per x). Applicando tale regola al Traveller, EURISKO sviluppò una nuova strategia: fece in modo che una nave danneggiata facesse fuoco su se stessa affondandosi. Poiché le regole convenzionali del gioco definivano l'agilità globale della flotta sulla base della nave più lenta, questo metodo era ragionevole per poter aumentare la potenza della flotta. Studiando la programmazione dei calcolatori, EURISKO prese, infine, in considerazione la funzione «x chiama y», dove x è un elemento del programma che attiva y, un altro elemento. La regola euristica dell'unione portò EURISKO a definire l'importante nozione di elementi ricorsivi di programmazione, componenti di un pezzo di software che richiamano se stessi.

«Fai coincidere» e «osserva i casi estremi» sono esempi di regole euristiche che guidano un programma di scoperta a definire nuovi concetti. Se si prende come missione del programma la ricerca di idee interessanti, ci deve essere anche un secondo tipo di euristica per aiutarlo a decidere quali dei molti concetti generati sono significativi. Le regole di sintesi dei concetti guidano la ricerca all'inizio; una volta avviatala, le regole euristiche di valutazione la incanalano lungo percorsi che vale la pena di esplorare. EURISKO ha anche regole come «Se tutti i membri di un insieme soddisfano in maniera inattesa qualche rara proprietà, allora aumenta il grado di interesse per quell'insieme e per l'euristica che ha portato alla sua definizione». Un'altra regola guida il programma, quando deve decidere quale studiare tra due concetti molto simili, a scegliere quello che richiede un minor tempo per il calcolatore o un minor numero di domande all'utente.

Dall'usare le regole euristiche per scoprire (o riscoprire) nuovi concetti o fatti all'usarle per scoprire nuove regole euristiche il passo teorico è breve. Quest'ultimo compito si collega a quello che è stato a lungo un obiettivo centrale della ricerca sull'intelligenza artificiale: scrivere programmi che apprendano dall'esperienza. In anni recenti, numerosi ricercatori hanno sviluppato programmi che traggono regole generali dalla loro esperienza nella risoluzione di singoli problemi. Il processo di generalizzazione è controllato dalla metaeuristica.

Il successo di DENDRAL spinse i suoi autori a scrivere un nuovo programma, METADENDRAL, che formula regole generali di spettrometria di massa, ricavate da osservazioni sul modo in cui particolari composti sono frammentati nello spettrometro. In questo caso, un esempio di regola metaeuristica è il semplice enunciato che le caratteristiche di una molecola più importanti per determinare il tipo di frammentazione sono quelle vicine ai punti di rottura. Applicando questa regola euristica, METADENDRAL potrebbe formulare una regola secondo cui le molecole organiche tendono a rompersi là dove gli atomi di carbonio e di ossigeno sono uniti da legami singoli. La nuova regola euristica servirebbe poi a dedurre la struttura di molecole sconosciute dai loro spettri di massa. Analogamente, Thomas M. Mitchell e Paul E. Utgoff della Rutgers University hanno scritto un programma chiamato LEX 2 che deriva regole euristiche per la soluzione di problemi nel calcolo integrale dalla sua esperienza nel calcolo di particolari integrali.

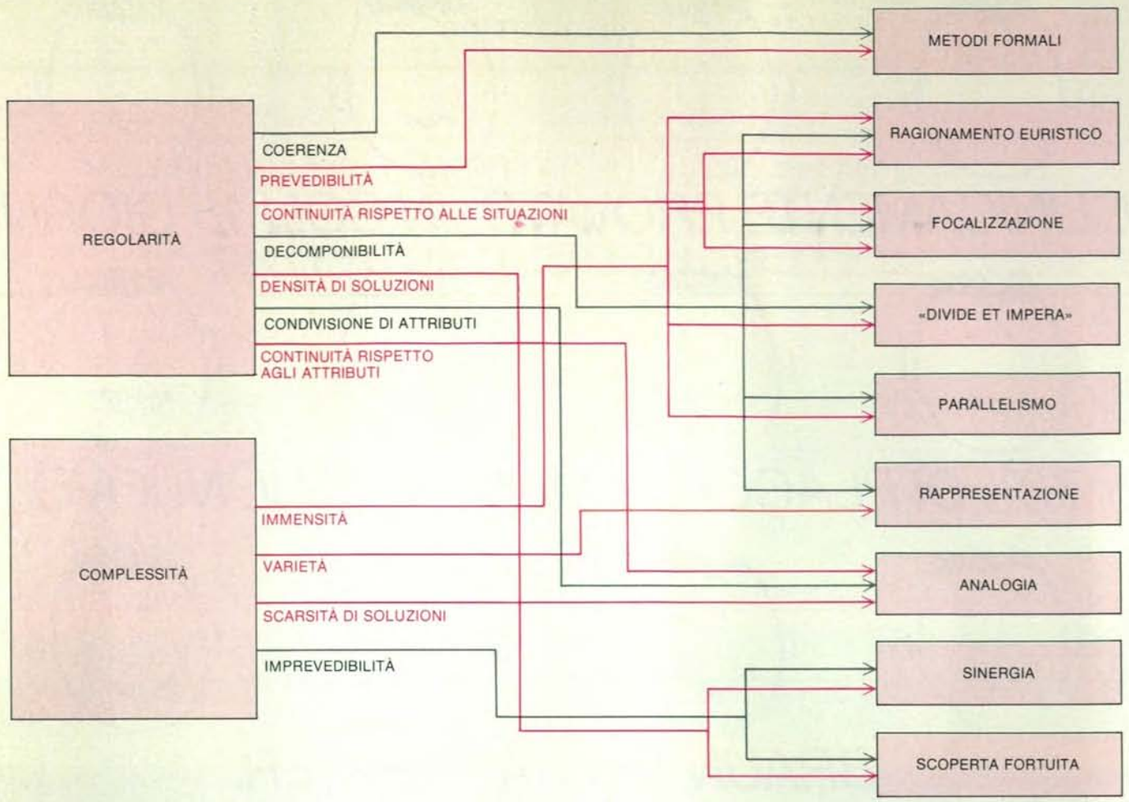
La progettazione di programmi ad apprendimento migliori dipende in parte dalla possibilità di trovare modi per sfruttare una fonte di potenza che sta veramente al centro dell'intelligenza umana: la capacità di capire e ragionare per analogia. Basta un minimo di introspezione e un ascolto attento per rendersi conto che ricorriamo continuamente all'analogia per spiegare e capire concetti e per trovarne di nuovi. Il software intelligente ha appena iniziato lo sfruttamento di questa fonte di potenza, ma essa sarà indubbiamente al centro della ricerca futura.

Non intendo dire con questo che finora non sia stato compiuto alcun progresso. Vent'anni fa Thomas G. Evans del Massachusetts Institute of Technology scrisse un programma capace di riconoscere analogie tra figure geometriche, un tipo di abilità richiesto per risolvere certi pro-

«FRED ASSOMIGLIA A UN ORSO»

NOME	FRED		NOME	ORSOCOMUNE
IDENTITÀ	UOMO		IDENTITÀ	ORSO
DIMORA	15 MAIN ST.		DIMORA	CAVERNA
TAGLIA	ROBUSTA	←	TAGLIA	ROBUSTA
ANDATURA	GOFFA	←	ANDATURA	GOFFA
CIBO	MIELE	←	CIBO	MIELE
ETÀ			ETÀ	5
UNGHIE	LUNGHE	← - - -	ARTIGLI	LUNGHI

I casellari *(frame)* sono un modo di rappresentare la conoscenza di un particolare concetto; tra i vantaggi che offrono, c'è quello di facilitare la deduzione di analogie da parte di un programma intelligente. Un casellario consiste di caselle riempite da attributi e da valori a essi associati. Se due oggetti hanno alcuni nomi di attributi identici, si può stabilire un'analogia semplicemente riempiendo caselle vuote di un casellario con appropriati valori dell'altro casellario. Regole euristiche guidano il programma nel determinare quali valori trasferire, conducendolo a considerare, per esempio, proprietà estreme del casellario di partenza. Quando una casella presente nel casellario di partenza è assente nell'altro, il programma può scegliere una casella analoga.



Le fonti di potenza nella risoluzione di un problema assumono significato *(linee in grigio)* e reale utilità o convenienza *(linee in colore)* grazie a certe proprietà del campo del problema. Per esempio, è possibile applicare il ragionamento euristico o la strategia *divide et impera* se il problema è regolare, nel senso che può essere scomposto in sottoproblemi. Tuttavia, è conveniente farlo solo se il campo è complesso e immenso; se è più limitato e regolare, può essere meglio ricorrere a un'impostazione logico-formale. Analogamente, si possono più rapidamente dedurre analogie in un campo (come quello della diagnosi medica) in cui gli oggetti (le malattie) hanno molti nomi di attributi in comune. Ragionare per analogia è conveniente solo quando c'è una continuità di valori degli attributi (malattie con sintomi e cause analoghe spesso richiedono un trattamento analogo) e quando un problema ha poche soluzioni (un insieme di sintomi è collegato a poche malattie).

blemi dei test d'intelligenza. Più difficile è far trovare ai programmi analogie concettuali e un certo numero di ricercatori sta lavorando a questo problema. Jaime G. Carbonell della Carnegie-Mellon University ha un programma che riconosce la somiglianza tra due algoritmi scritti in due diversi linguaggi per calcolatore ma aventi lo stesso scopo. EURISKO, da parte sua, non tanto trova analogie quanto piuttosto utilizza un ragionamento analogico di basso livello. Lavorando sulla progettazione di circuiti integrati, per esempio, ha trovato che la simmetria è una proprietà auspicabile per i chip, senza però capire perché; quando in seguito gli è stato insegnato a progettare flotte per il Traveller, ha deciso di farle simmetriche e ha giustificato la sua decisione facendo riferimento alla sua precedente esperienza nella progettazione di circuiti.

Si tratta, tuttavia, di una capacità straordinariamente misera se paragonata a quelle umane. La scadente prestazione dei programmi per calcolatore nella scoperta e nell'uso delle analogie può essere attribuibile più alla limitatezza delle conoscenze che all'incapacità dei programmatori di trovare algoritmi adeguati. Gli esseri umani hanno un enorme magazzino di concetti a cui attingere come possibili analoghi: forse un milione di ricordi di oggetti, di azioni, di emozioni, di situazioni e via dicendo. Il software attualmente esistente non ha questo ricco repertorio, né i programmi hanno la possibilità di accumulare un grosso insieme di esperienze da cui trarre confronti. I programmi, che girano a lungo prima di essere fermati e fatti ripartire, di solito non tengono adeguate registrazioni della loro esperienza di ricerca e, quando vengono fermati, perdono tutto o la maggior parte di ciò che hanno imparato. Anche EURISKO, che ha girato per settimane di seguito ed è ripartito con la maggior parte delle sue registrazioni intatte, ha avuto una breve vita mentale, con esperienze che, per varietà, non si avvicinano nemmeno a quelle di un essere umano neonato.

La ricetta per migliorare il ragionamento analogico di un programma è quindi uguale a quella per migliorare il rendimento generale del software intelligente: espandere la base di conoscenza. Idealmente si potrebbe immagazzinare un'intera enciclopedia in forma accessibile al calcolatore, non come testo ma come raccolta di migliaia di unità strutturate e dotate di molteplici indici. Il lavoro preliminare svolto in questa direzione da alcuni ricercatori ha rivelato che la cosa è ancora più difficile di quanto sembri: la comprensione degli articoli di enciclopedia richiede essa stessa un ampio corpo di conoscenze, legate al senso comune, che il software dei calcolatori non possiede ancora.

Da una parte, i programmi per calcolatore dovranno diventare ben più ricchi di conoscenze prima di essere in grado di ragionare efficacemente per analogia. Dall'altra, per acquisire un simile carico di conoscenze i calcolatori dovrebbero essere almeno in grado di «capire» l'analogia quando si presenta e certo questa è una delle più potenti tecniche d'apprendimento di cui dispongano gli esseri umani. È un poco il problema dell'uovo e della gallina. Fortunatamente, per un calcola-

Strutture di dati e algoritmi

Sono i costituenti essenziali di ogni programma per calcolatore: la scelta delle strutture di dati e le procedure per manipolarle contengono la chiave per verificare che un programma faccia ciò per cui è creato

di Niklaus Wirth

Le strutture di dati e gli algoritmi sono i materiali impiegati nella costruzione di programmi. Lo stesso calcolatore non consiste di altro che di strutture di dati e algoritmi; le strutture di dati cablate sono i registri e le parole di memoria che contengono valori codificati in codice binario; gli algoritmi sono le regole fisse, incorporate nei circuiti logici elettronici, secondo le quali i dati memorizzati sono interpretati come istruzioni da eseguire. Pertanto al livello fondamentale un calcolatore può lavorare con un solo tipo di dati, i singoli bit o cifre binarie, e può operare sugli stessi secondo un unico insieme di algoritmi, quelli definiti dall'insieme di istruzioni dell'unità centrale.

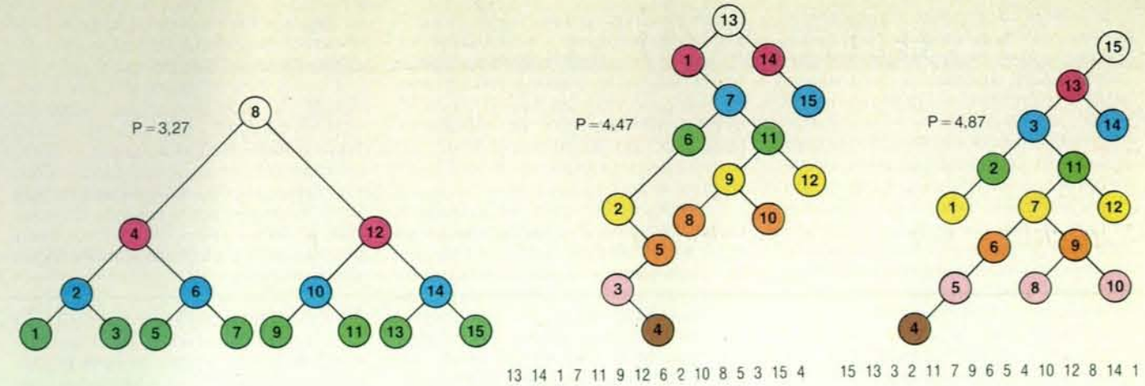
I problemi che comunemente si vogliono risolvere con l'aiuto dei calcolatori sono raramente espressi in termini di bit; i dati hanno piuttosto la forma di numeri, caratteri, testi, eventi, simboli e strutture più complesse come sequenze, liste e alberi. Gli algoritmi impiegati nella soluzione dei problemi sono anche più variati: esistono infatti almeno tanti algoritmi quanti sono i problemi computazionali. Com'è possibile risolvere una vasta gamma di problemi con una macchina che opera sempre secondo regole fisse? La spiegazione è che il calcolatore è un vero dispositivo di uso generale *(general-purpose)*, cioè adatto a molti impieghi, la cui natura può essere interamente trasformata dal programma in esecuzione. Il principio basilare fu enunciato per la prima volta da John von Neumann: un certo insieme di informazioni può costituire in un certo istante dei dati sui quali opera un programma e all'istante successivo la stessa informazione può essere interpretata come un programma a tutti gli effetti. Un programma può quindi essere formulato in termini di nozioni familiari, consone al problema da risolvere; un altro programma, chiamato assemblatore o compilatore, traduce poi queste nozioni in termini delle operazioni disponibili nel calcolatore.

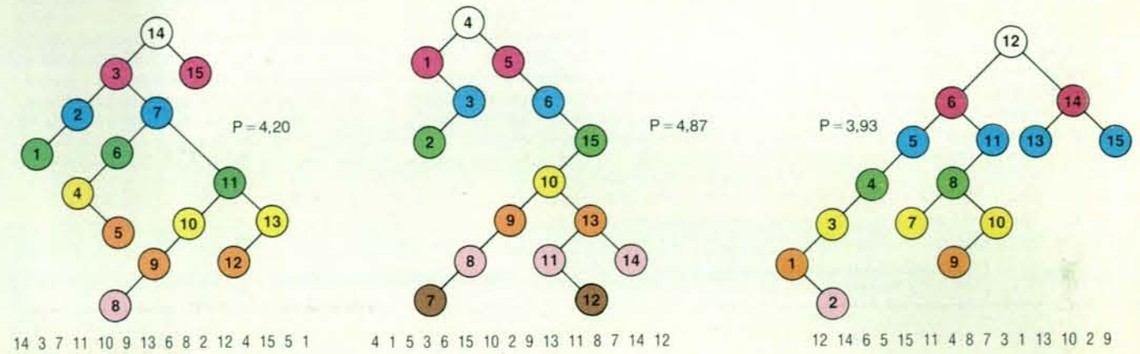
In questo modo è possibile costruire sistemi di straordinaria complessità. Il programmatore stabilisce una gerarchia di astrazioni, vedendo il programma prima a grandi linee e quindi occupandosi di una parte alla volta, ignorando momentaneamente i dettagli delle altre parti. Il processo di astrazione non è una pura convenzione: è una necessità, poiché programmi di dimensione appena più che banale non potrebbero venir scritti se si dovesse lavorare con una massa omogenea e non differenziata di bit. Senza astrazioni di più alto livello un programma non potrebbe venir compreso pienamente neppure dal suo autore.

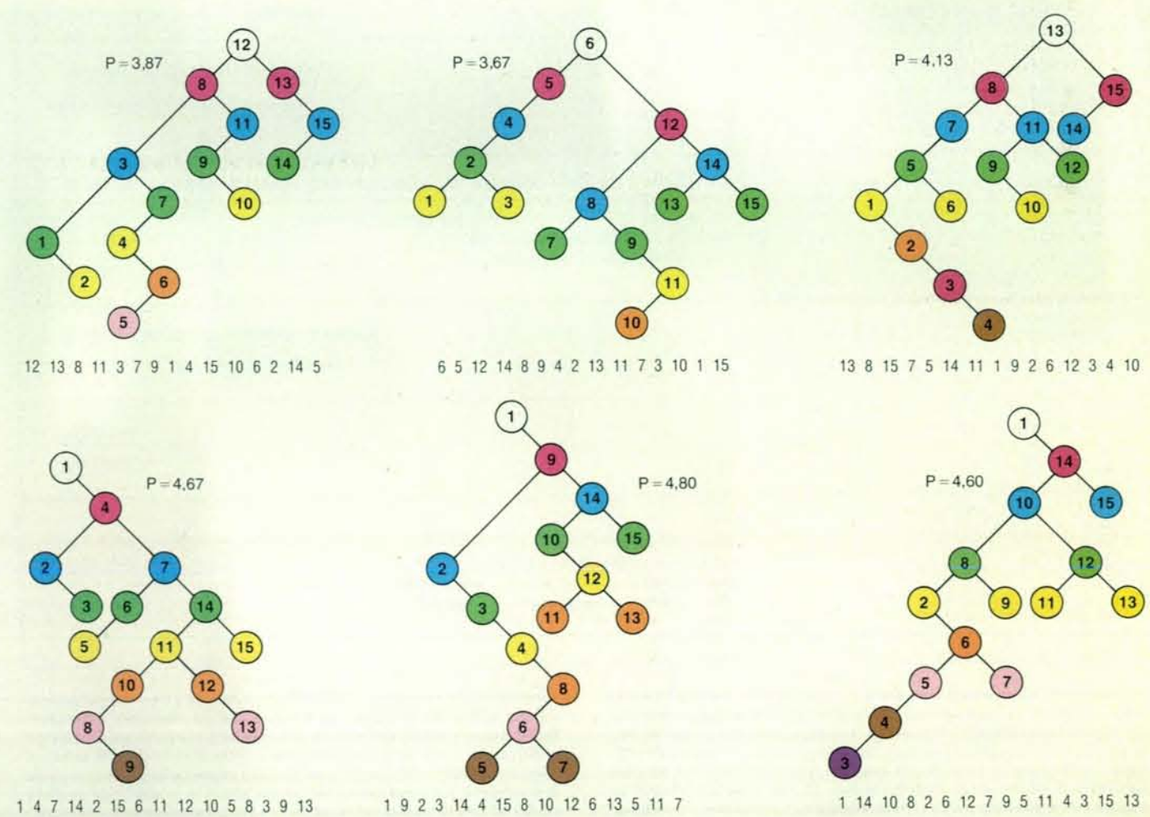
Per specificare le strutture di dati astratte e gli algoritmi di un programma è necessaria una notazione formale, nella quale il significato di ogni istruzione legale sia definito con precisione e in modo non ambiguo. Tali notazioni formali per la programmazione sono note come linguaggi, ma il termine è fuorviante perché programmare somiglia solo superficialmente a scrivere. Io preferisco pensare alla programmazione come all'attività di progettare una nuova macchina (da realizzarsi mediante una macchina esistente adatta a differenti scopi). Il progetto è specificato in termini delle possibilità espressive della notazione, esattamente come un dispositivo elettronico viene disegnato mediante i simboli dei circuiti di base e delle loro connessioni. Se si interpreta la programmazione come il progetto di una macchina, la necessità di precisione diviene quanto mai ovvia.

Tra le possibilità fornite da quasi tutti i linguaggi di programmazione vi è la facoltà di riferirsi a un dato assegnandogli un nome o identificatore. Alcune delle entità che ricevono un nome sono costanti che mantengono lo stesso valore in tutto il segmento di programma nel quale sono definite; per esempio si potrebbe assegnare a *pi* il valore 3.14159. Altre entità con nome sono variabili, alle quali si possono assegnare nuovi valori mediante istruzioni all'interno del programma, cosicché il loro valore non è noto sino all'esecuzione del programma. Le variabili *diametro* e *circonferenza* potrebbero assumere differenti valori ogni volta che viene eseguito un calcolo.

Una «foresta» di alberi binari illustra la stretta relazione tra strutture di dati e algoritmi. Un albero binario, che per convenzione cresce dalla radice verso il basso, è una struttura di dati scelta frequentemente quando degli oggetti debbono venir ritrovati in ordine casuale in un insieme ampio. L'albero è costituito da nodi identificati dal valore di una «chiave»; nei diagrammi della pagina a fronte la chiave è un intero da 1 a 15. Ogni nodo ha al massimo due «figli» o sottonodi, ordinati in modo che il figlio sinistro abbia sempre una chiave inferiore a quella del padre e il nodo destro una superiore. L'albero ottimale è quello in alto a sinistra: è perfettamente bilanciato; pertanto il numero medio di nodi da attraversare per raggiungere un nodo dato è ridotto al minimo. (La lunghezza del cammino per raggiungere ciascun nodo è indicata dal colore, secondo la legenda mostrata sopra.) Gli altri alberi sono stati generati da un algoritmo di inserimento che attacca un nodo nella prima posizione legale trovata, senza spostare alcun altro nodo per mantenere il bilanciamento dell'albero. Un algoritmo più sofisticato potrebbe ridurre leggermente il cammino medio, ma sarebbe esso stesso più complesso. L'algoritmo di inserimento e i benefici del bilanciamento vengono esaminati nelle illustrazioni delle pagine 36 e 37. L'ordine di inserimento dei nodi è indicato sotto ogni albero. Il cammino medio è indicato a fianco di ogni albero.







Il nome di una costante o di una variabile è un aiuto mnemonico per il programmatore, ma non ha alcun significato per il calcolatore: il compilatore che traduce il testo di un programma in codice binario associa semplicemente ogni identificatore con un indirizzo in memoria. Se un'istruzione richiede la moltiplicazione di *diametro* per *pi*, il calcolatore estrae i valori memorizzati agli indirizzi corrispondenti e ne calcola il prodotto; se il risultato deve diventare il nuovo valore di *circonferenza*, viene memorizzato all'indirizzo corrispondente a tale etichetta.

Dare nomi a costanti e a variabili nella programmazione è simile all'uso di espressioni simboliche nell'algebra, ma perché un calcolatore sia in grado di svolgere il suo compito debbono venirgli fornite alcune ulteriori informazioni che individuino il «tipo» delle entità con un nome. Una persona che risolva manualmente un problema ha una comprensione intuitiva [derivata peraltro dall'esperienza, *N. d. T.*] dei tipi di dato e delle operazioni valide per ogni tipo; è noto, per esempio, che non si può estrarre la radice quadrata di una parola o trascrivere in maiuscolo un numero. Uno dei motivi per i quali tali distinzioni vengono facilmente fatte è che parole, numeri e vari altri simboli vengono rappresentati in modo alquanto differente. Per un calcolatore, però, tutti i tipi di dati vengono rappresentati mediante sequenze di bit e le distinzioni di tipo vanno rese esplicite.

Supponiamo che nello svolgimento di alcune operazioni il valore binario di sette bit 1010011 sia stato letto in un registro nell'unità centrale di elaborazione di un calcolatore: come dev'essere interpretato tale valore? Una possibilità è che rappresenti un numero cardinale, nel qual caso l'equivalente in notazione decimale sarebbe 83. In molti linguaggi di programmazione tale valore potrebbe anche rappresentare un intero con segno, equivalente a −45 in notazione decimale. Gli stessi bit potrebbero codificare non un numero, ma un carattere; secondo l'American Standard Code for Information Interchange (ASCII) la codifica binaria 1010011 rappresenta la lettera *S*. Tra le altre possibilità, il codice binario potrebbe rappresentare non un dato, bensì un'istruzione: la sua interpretazione dipenderebbe allora dal calcolatore.

I tipi di dato utilizzati dai comuni linguaggi di programmazione includono numeri cardinali, interi, numeri reali (approssimati mediante frazioni), insiemi, caratteri e stringhe di caratteri. L'informazione sul tipo di ogni variabile non serve solamente per la corretta interpretazione della codifica binaria, ma anche per riservare un'adeguata quantità di spazio in memoria.

In alcuni linguaggi di programmazione il compilatore deduce il tipo di una costante o di una variabile da certe indicazioni su come è scritto il valore assegnato; la presenza di un punto decimale, per esempio, potrebbe indicare un numero reale. Altri linguaggi richiedono invece che il programmatore dichiari esplicitamente il tipo di ogni variabile. La dichiarazione esplicita dei tipi di dato può sembrare noiosa, se può essere evitata, ma ha un importante vantaggio: sebbene il valore di una variabile possa cambiare ripetutamente durante l'esecuzione di un programma, il suo tipo non dovrebbe mai cambiare; il compilatore può pertanto eseguire dei controlli per verificare che tutte le operazioni eseguite su una variabile siano coerenti con la dichiarazione di tipo. Tali controlli di coerenza possono essere effettuati esaminando il testo del programma e pertanto restano validi per tutti i possibili calcoli specificati dal programma. Viceversa un'esecuzione di prova del programma compilato può verificare la correttezza delle operazioni solo per gli specifici valori di ingresso provati.

La nozione di tipi di dato è stata estesa, principalmente attraverso il linguaggio di programmazione Pascal, a comprendere la descrizione di strutture di dati. Una variabile strutturata è formata da una molteplicità di elementi, o componenti, ma ciononostante ci si può riferire a essa come a una singola entità. In un calendario, per esempio, si deve poter specificare una singola data, ma deve anche esistere un modo per riferirsi a mesi e ad anni interi. La dichiarazione di tipo per una variabile strutturata fissa il numero di elementi che la costituiscono, permettendo al compilatore di riservare la memoria necessaria, e fornisce informazioni sul metodo con cui si intende accedere ai singoli elementi.

Se tutti gli elementi sono dello stesso tipo (e pertanto richiedono la stessa quantità di memoria), la variabile strutturata è detta omogenea e può essere dichiarata come vettore, ossia matrice *(array)* unidimensionale. Una variabile strutturata cui si vuol dare il nome *Sett* e consistente di 30 numeri cardinali può avere la seguente dichiarazione di tipo:

Sett: array [1..30] of cardinal

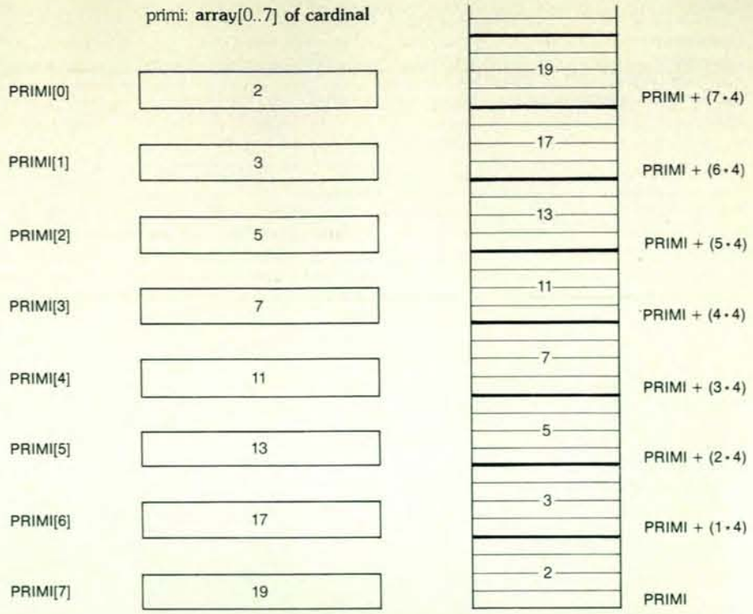
In un vettore i singoli elementi sono identificati per mezzo dell'indice, che determina la posizione nella sequenza di elementi. L'indirizzo del quinto elemento, per esempio, è semplicemente l'indirizzo del primo elemento più quattro volte la dimensione di un elemento.

Se gli elementi di una variabile strutturata non sono tutti dello stesso tipo, questa semplicità di accesso si perde. D'altra parte elementi che differiscono per il tipo è probabile che differiscano anche per altre caratteristiche e vi è meno convenienza a farvi riferimento per mezzo di un indice; conviene invece dare a ogni elemento un proprio identificatore. L'intera variabile strutturata è detta *record* e gli elementi vengono chiamati campi. Un record costruito per contenere informazioni su città è mostrato nell'illustrazione in basso di questa pagina; il riferimento al record avviene mediante il nome di variabile *citta;* singoli campi sono designati come *citta.nome, citta.popolazione* e così via. (In Pascal, negli identificatori non è ammesso l'uso di lettere accentate o dell'apice come accento.)

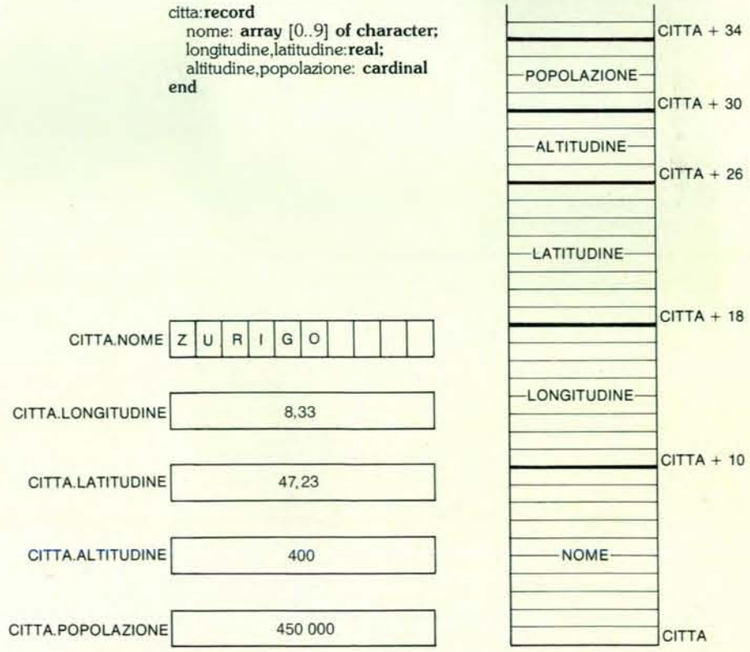
Un altro tipo di struttura di dati fondamentale è l'insieme. È utile quando il valore di un elemento non è di immediato interesse e conta solo la sua presenza o assenza. Se una variabile di nome *primi* è dichiarata come insieme di numeri

TIPO	RAPPRESENTAZIONE ESTERNA	RAPPRESENTAZIONE INTERNA
CARDINALE	83	00000000 00000000 00000000 01010011 VALORE
INTERO	− 83	1 1111111 1111111 1111111 10101101 SEGNO VALORE IN COMPLEMENTO A DUE
REALE	83,0	0 1000111 10100110 00000000 00000000 00000000 00000000 00000000 00000000 SEGNO DELLA MANTISSA ESPONENTE MODIFICATO MANTISSA NORMALIZZATA
INSIEME	0, 1, 4, 6	00000000 00000000 00000000 01010011 MEMBRI DELL'INSIEME
CARATTERE	S	01010011 CODICE ASCII
STRINGA	SET 83	S E T <SPAZIO> 8 3 00000110 01010011 01000101 01010100 00100000 00111000 00110011 NUMERO DI CARATTERI CODICE ASCII DEI CARATTERI

Ai tipi di dati elementari viene associata una rappresentazione interna predeterminata dal compilatore che traduce le istruzioni di un linguaggio di programmazione. Il tipo di ogni variabile deve essere specificato, in modo che il compilatore possa assegnare spazio di memoria e codificare i dati in modo corretto. In queste rappresentazioni un numero cardinale o intero viene codificato con 32 bit, o quattro byte; un numero reale richiede 64 bit e viene codificato in notazione scientifica, con esponente, mantissa e segno. Un insieme può essere rappresentato da una stringa di bit, dove un 1 indica che un elemento è membro dell'insieme e uno 0 che non lo è. I caratteri vengono generalmente codificati in 7 o 8 bit secondo il codice ASCII. Una stringa di caratteri consiste dei codici dei caratteri più un byte supplementare che contiene la lunghezza. La suddivisione dei valori binari in gruppi è un artificio grafico al fine di migliorare la leggibilità e non compare nella memoria.



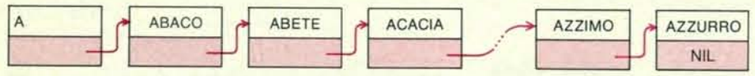
Un vettore, ossia una matrice unidimensionale, consiste in un numero fissato di elementi, tutti dello stesso tipo di dato fondamentale; il compilatore può assegnare a ognuno la stessa quantità di memoria e quindi un elemento può essere ritrovato con un semplice calcolo dell'indirizzo utilizzando il valore dell'indice. Qui, a un vettore di otto elementi è stato dato il nome *primi* che rappresenta anche l'indirizzo di inizio del vettore in memoria; gli elementi sono numeri cardinali che richiedono quattro byte di memoria, cosicché l'indirizzo di un qualsiasi elemento del vettore può essere calcolato moltiplicando l'indice dell'elemento per 4 e sommando l'indirizzo di *primi*.



Un *record* contiene informazioni eterogenee. Nell'esempio è stato definito un record, chiamato *citta*, che include cinque campi: una stringa di caratteri, due numeri reali e due numeri cardinali. (Diamo per scontato qui che siano stati definiti i tipi *character, real* e *cardinal* secondo il significato intuitivo.) Si può accedere a un singolo campo mediante il nome del record e il nome del campo separati da un punto. Dato che differenti tipi di dati richiedono differenti quantità di memoria, i campi non sono tutti della stessa lunghezza; per ognuno il compilatore deve tener conto dello spostamento: la distanza in memoria tra l'inizio dell'intero record e l'inizio del campo.

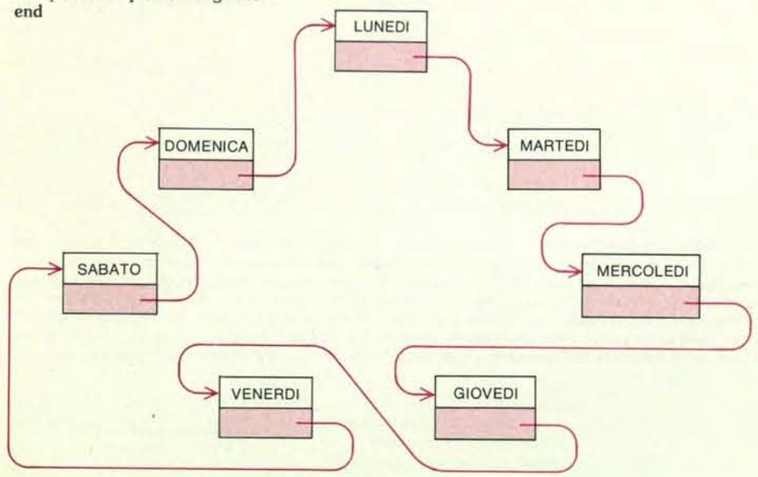
LISTA

```
type parola =
  record
    grafia: array[0..20] of character;
    prossimo: pointer to parola
end
```



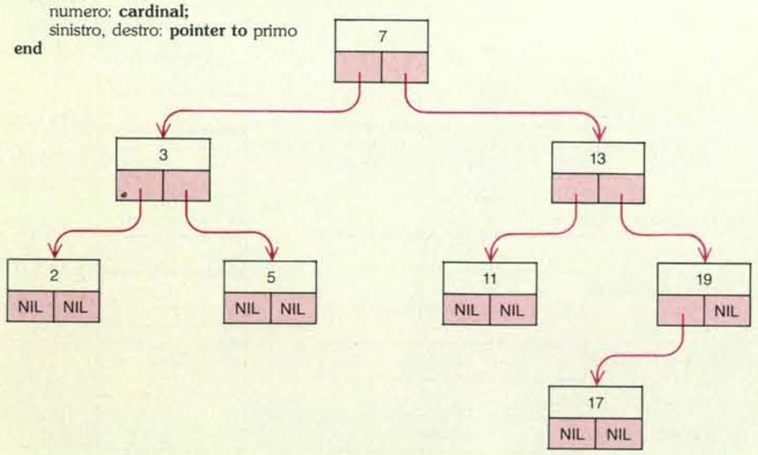
LISTA CIRCOLARE

```
type giorno =
  record
    quale: (LUNEDI, MARTEDI, MERCOLEDI, GIOVEDI, VENERDI,
      SABATO, DOMENICA);
    prossimo: pointer to giorno
end
```



ALBERO

```
type primo =
  record
    numero: cardinal;
    sinistro, destro: pointer to primo
end
```



Le strutture di dati dinamiche possono espandersi o contrarsi o persino riorganizzarsi sotto il controllo della parte algoritmica del programma; sono costituite da nodi, generalmente record, che includono puntatori ad altri nodi *(in colore)*; un puntatore *(pointer)* che non punta a nulla ha il valore speciale *nil*. La struttura dinamica più semplice è la lista concatenata, in cui ogni nodo contiene un puntatore al successivo: l'esempio in figura potrebbe costituire parte di un dizionario. Si può far diventare una lista circolare facendo sì che l'ultimo puntatore punti al primo elemento. In un albero binario ogni nodo ha due puntatori che danno gli indirizzi dei sottonodi sinistro e destro.

cardinali, è possibile definire un'operazione di esame di presenza che produca il valore logico «vero» se un numero è nell'insieme e «falso» altrimenti. Gli insiemi possono essere realizzati e manipolati in modo efficiente: ogni elemento dell'insieme è rappresentato da un singolo bit, indicando con il valore 1 la presenza dell'elemento e con 0 la sua assenza.

Vettori, record e insiemi sono chiamati strutture di base; in molti contesti sono necessarie strutture più complicate, ma anziché tentare di inventare una notazione per ognuna di esse è preferibile introdurre un metodo generale per costruire strutture arbitrarie. Mentre la struttura di un vettore, di un record o di un insieme resta costante durante l'esecuzione di un programma, le strutture più complesse possono crescere, ridursi o alterare la loro topologia. Le strutture di base sono statiche, quelle derivate sono dinamiche.

Dato che la dimensione di una struttura dinamica è soggetta a cambiamenti, non può essere specificata da una dichiarazione invariante, ma deve essere definita nella parte algoritmica del programma. Per la stessa ragione deve essere il programma stesso, e non il compilatore, ad assegnare spazio di memoria per la struttura. Questa sembrerebbe essere una situazione pericolosa, poiché la correttezza della struttura non può essere verificata durante la compilazione. Una proprietà della struttura rimane tuttavia fissa e può pertanto essere dichiarata in anticipo, vale a dire i tipi degli elementi dai quali la struttura è composta. Solo il numero degli elementi e le loro connessioni possono variare durante l'esecuzione.

Il meccanismo per la creazione di strutture dinamiche consiste in un modo per generare componenti di base, chiamati nodi, e in un modo per stabilire delle connessioni tra i nodi. I nodi sono generalmente record e le connessioni sono definite da variabili chiamate puntatori *(pointer)*. Come il nome stesso suggerisce, un puntatore punta a un elemento della struttura dinamica; a un puntatore può anche essere assegnato il valore speciale *nil*, nel qual caso non punta a nulla.

Una struttura dinamica che può essere creata facilmente mediante nodi e puntatori è una lista concatenata: ogni elemento della lista è un record, di cui uno dei campi è un puntatore al record seguente; il puntatore contenuto nell'ultimo record ha valore *nil*. Per aggiungere un record il programma predispone lo spazio necessario in memoria e cambia l'ultimo puntatore della lista in modo che punti al nuovo elemento. Se l'ultimo elemento punta al primo, la lista viene convertita in un anello. Un albero può venir creato includendo in ogni nodo i puntatori a tutti i suoi sottonodi. Esempi di parecchie strutture dinamiche sono mostrati nell'illustrazione di questa pagina.

La realizzazione dei puntatori è immediata: il valore di un puntatore (se non è *nil*) è l'indirizzo del nodo cui punta. Perché allora non chiamare semplicemente indirizzo un puntatore? È opportuno conservare la distinzione perché per un puntatore si dichiara che punta a una variabile di tipo noto, mentre un indirizzo potrebbe specificare qualunque locazione in memoria. Il compilatore può pertanto accertarsi che ogni puntatore sia associato a un oggetto del tipo corretto.

Una struttura di dati è un concetto essenzialmente spaziale: è riducibile a una mappa dell'organizzazione dell'informazione nella memoria del calcolatore. Un algoritmo è il corrispondente elemento procedurale nella struttura di un programma: è una ricetta per il calcolo.

I primi algoritmi furono inventati per risolvere problemi numerici, come moltiplicare numeri, trovare il massimo comune divisore, calcolare funzioni trigonometriche e così via. Oggi gli algoritmi non numerici sono di pari importanza; sono stati sviluppati per operazioni come trovare l'elemento minimo in una sequenza, cercare una parola data in un testo, pianificare attività e classificare dati secondo qualche specifico ordinamento.

Gli algoritmi non numerici operano su dati che non sono necessariamente numeri; inoltre per idearli o capirli non sono necessari profondi concetti matematici. Non è però lecito dedurne che la matematica non abbia alcun ruolo nello studio di tali algoritmi: al contrario, metodi matematici rigorosi sono essenziali per trovare la migliore soluzione a problemi non numerici, per dimostrare la correttezza delle soluzioni e per determinarne l'efficacia. La programmazione resta una disciplina altamente matematica; sono però stati scambiati i ruoli: mentre metodi basati sul calcolo erano una volta impiegati per risolvere problemi matematici, metodi matematici sono ora applicati alla soluzione di problemi di calcolo.

Non tenterò in questa sede di dare una panoramica generale delle varie categorie di algoritmi o di discutere metodi per la costruzione di nuovi algoritmi; cercherò piuttosto di illustrare l'impostazione moderna del ragionamento sugli algoritmi. Dato un algoritmo che mira a risolvere un problema, come si può comprenderlo e, in particolare, acquistare fiducia sulla sua correttezza senza ricorrere a un calcolatore per «provare alcuni casi»?

Se un algoritmo è visto come una serie temporale di operazioni, una questione fondamentale è come venga controllato il flusso delle operazioni. Quando l'esecuzione di un programma ha raggiunto una particolare istruzione, come determina il calcolatore quale sia la prossima istruzione da eseguire?

È stato dimostrato che tre soli principi di controllo sono sufficienti per descrivere qualunque algoritmo. Il primo principio è talmente ovvio che viene spesso trascurato: è la nozione di sequenza; a meno che il calcolatore sia istruito diversamente, esegue le istruzioni di un programma sequenzialmente. Il secondo principio è l'esecuzione condizionale che è generalmente designata nel testo del programma da un costrutto «*if...then*» («se...allora»). Nell'istruzione «*if B then S*», *B* è un'espressione booleana, che può produrre soltanto i valori vero o falso, e *S* è una

```
a := x; b := y; c := 0;
while b ≠ 0 do
    {asserzione: a*b + c = x*y}
    {asserzione: b≠ 0}
    b := b - 1; c := c + a
end
    {asserzione: a*b + c = x*y e b = 0 implica c = x*y}
```

OPERAZIONI	VALORI	INVARIANTE DI CICLO	GUARDIA
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 · 13 + 0 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 12, c = 7	7 · 12 + 7 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 11, c = 14	7 · 11 + 14 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 10, c = 21	7 · 10 + 21 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 9, c = 28	7 · 9 + 28 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 8, c = 35	7 · 8 + 35 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 7, c = 42	7 · 7 + 42 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 6, c = 49	7 · 6 + 49 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 5, c = 56	7 · 5 + 56 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 4, c = 63	7 · 4 + 63 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 3, c = 70	7 · 3 + 70 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 2, c = 77	7 · 2 + 77 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 1, c = 84	7 · 1 + 84 = 7 · 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 0, c = 91	7 · 0 + 91 = 7 · 13	b = 0

```
a := x; b := y; c := 0;
while b ≠ 0 do
    {asserzione: a*b + c = x*y}
    {asserzione: b≠ 0}
    c := c + a*(b mod 10);
    a := 10*a;
    b := b div 10
end
    {asserzione: a*b + c = x*y e b = 0 implica c = x*y}
```

OPERAZIONI	VALORI	INVARIANTE DI CICLO	GUARDIA
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 · 13 + 0 = 7 · 13	b ≠ 0
c := c + a · (b mod 10) a := 10 · a b := b div 10	c = 0 + 7 · 3 = 21 a = 10 · 7 = 70 b = 13 div 10 = 1	70 · 1 + 21 = 7 · 13	b ≠ 0
c := c + a · (b mod 10) a := 10 · a b := b div 10	c = 21 + 70 · 1 = 91 a = 10 · 70 = 700 b = 1 div 10 = 0	700 · 0 + 91 = 7 · 13	b = 0

```
a := x; b := y; c := 0;
while b ≠ 0 do
    {asserzione: a*b + c = x*y}
    {asserzione: b≠ 0}
    if Odd (b) then c := c + a e;
    a := 2*a;
    b := b div 2
end
    {asserzione: a*b + c = x*y e b = 0 implica c = x*y}
```

OPERAZIONI	VALORI	INVARIANTE DI CICLO	GUARDIA
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 · 13 + 0 = 7 · 13	b ≠ 0
if Odd (b) then c := c + a end a := 2 · a b := b div 2	c = 0 + 7 = 7 a = 2 · 7 = 14 b = 13 div 2 = 6	14 · 6 + 7 = 7 · 13	b ≠ 0
if Odd (b) then c := c + a end a := 2 · a b := b div 2	c = 7 a = 2 · 14 = 28 b = 6 div 2 = 3	28 · 3 + 7 = 7 · 13	b ≠ 0
if Odd (b) then c := c + a end a := 2 · a b := b div 2	c = 7 + 28 = 35 a = 2 · 28 = 56 b = 3 div 2 = 1	56 · 1 + 35 = 7 · 13	b ≠ 0
if Odd (b) then c := c + a end a := 2 · a b := b div 2	c = 35 + 56 = 91 a = 2 · 56 = 112 b = 1 div 2 = 0	112 · 0 + 91 = 7 · 13	b = 0

Lo sviluppo di un algoritmo per moltiplicare due numeri cardinali procede attraverso tre fasi. Il primo algoritmo *(in alto)* impiega il metodo delle addizioni ripetute; la sua correttezza può essere verificata tramite un'asserzione chiamata invariante di ciclo, che deve rimanere vera in ogni fase del calcolo. Un metodo più efficiente *(al centro)* è quello generalmente usato nell'eseguire manualmente la moltiplicazione; esso richiede di dividere il moltiplicatore per 10 invece di decrementarlo di 1; nell'esempio della moltiplicazione 7 * 13 questo algoritmo riduce il numero di esecuzioni del ciclo da 13 a 2. Poiché un calcolatore digitale usa l'aritmetica binaria, un algoritmo basato sulla divisione per 2 *(in basso)* è ancora più veloce, nonostante richieda più ripetizioni.

qualsiasi istruzione o gruppo di istruzioni. *B* viene valutata e *S* viene eseguita solo se il valore risultante è vero.

Il terzo principio è la ripetizione, che può essere indicata da un costrutto «*while...do*» («mentre...esegui»). «*While B do S*» esamina il valore di *B* e, se è vero, esegue *S*: i due passi vengono ripetuti finché una valutazione di *B* produce valore falso. Nella maggior parte dei casi un'istruzione in *S* provoca prima o poi il cambiamento del valore di *B*, in modo che il ciclo non continui indefinitamente. Sia nel costrutto condizionale sia in quello di ciclo, *B* ha la funzione di una «guardia», un'espressione che permette l'esecuzione di *S* solo se la condizione definita da *B* è soddisfatta.

Consideriamo un algoritmo per moltiplicare due numeri cardinali, x e y, per mezzo di ripetute addizioni; una descrizione formale dell'algoritmo è data nell'illustrazione della pagina precedente. Il primo passo consiste nel predisporre tre variabili: il moltiplicando a, il moltiplicatore b e una somma parziale c, che alla fine diverrà il prodotto. Alle variabili vengono assegnati i loro valori iniziali nelle tre istruzioni $a := x$, $b := y$ e $c := 0$. Il simbolo «:=» rappresenta qui l'operatore di assegnazione; mentre il segno di uguale costituisce un'asserzione di uguaglianza, l'operatore di assegnazione crea una condizione di uguaglianza, vale a dire assegna al simbolo a sinistra il valore dell'espressione a destra. Può anche essere letto come «Fai diventare a uguale a x».

Il cuore dell'algoritmo è un ciclo *while* nel quale la guardia è l'asserzione $b \neq 0$. Finché b rimane maggiore di zero, due operazioni vengono eseguite ripetutamente. Nella prima operazione b viene decrementato di 1; nella seconda a viene sommato al valore corrente di c. Queste operazioni vengono formalmente espresse nelle due istruzioni di assegnazione:

$$b := b - 1;\ c := c + a$$

L'intento dell'algoritmo è ovvio; con c inizialmente uguale a 0, a gli viene sommato b volte, cosa che realizza direttamente la definizione di moltiplicazione.

Come si può essere sicuri, tuttavia, che un programma scritto per esprimere questa idea intuitivamente chiara codifichi effettivamente l'algoritmo corretto? Una via possibile è quella di introdurre il programma in un calcolatore, compilarlo e verificare alcuni casi. Questo metodo non può mai portare all'assoluta fiducia nella correttezza del programma, semplicemente perché il numero delle prove possibili è infinito. Una risposta migliore consiste nell'includere nel programma «asserzioni» o definizioni di condizioni che debbono essere vere (se l'algoritmo è corretto) indipendentemente dal cammino percorso dalla computazione per giungere sino a quel punto. In questo caso l'asserzione è un «invariante di ciclo», un'affermazione che vale quale che sia il numero di volte per le quali il ciclo è stato eseguito.

Quale asserzione sulle entità in gioco nel problema della moltiplicazione rimane vera durante tutta l'esecuzione del programma? Dato che a e b sono inizialmente poste uguali a x e y, è chiaro che all'inizio $a * b$ deve essere uguale a $x * y$. (L'asterisco indica la moltiplicazione secondo una convenzione comune a molti linguaggi di programmazione.) Analogamente quando il calcolo è terminato, c viene preso come prodotto e pertanto anche c deve essere uguale a $x * y$. Nei vari stadi intermedi tra l'inizio e la fine della procedura, b viene decrementato di 1 ogni volta che c è incrementato di a. Da questa analisi segue che l'equazione $a * b + c = x * y$ vale durante tutto il calcolo; tale asserzione è pertanto l'invariante essenziale nel ciclo *while;* insieme alla condizione di arresto ($b = 0$), prova il risultato desiderato ($c = x * y$).

In questo caso la veridicità dell'asserzione impiegata per confermare la correttezza del programma è appena più ovvia della correttezza delle istruzioni stesse del programma. L'algoritmo scelto, tuttavia, è semplice. La potenza delle asserzioni come mezzo per verificare la correttezza dei programmi diviene evidente quando l'algoritmo è raffinato per renderlo più efficiente. L'asserzione formulata nel caso più semplice resta valida anche se il programma diviene più complicato.

Nell'algoritmo, il ciclo di moltiplicazione per addizione ripetuta deve essere eseguito y volte. Esistono però metodi più veloci. L'algoritmo per la moltiplicazione insegnato alle elementari ne è un esempio; è basato sullo stesso principio, ma b viene ridotto a passi più ampi. Invece di essere decrementato di 1, viene diviso per 10, un'operazione particolarmente facile nel sistema decimale. Di fatto in questo contesto non si pensa neppure al processo come a una divisione, ma piuttosto come alla scomposizione del moltiplicatore nelle cifre che lo compongono. L'isolamento di cifre può essere effettuato in termini algoritmici applicando due operatori matematici, DIV e MOD, che producono rispettivamente la parte intera del quoziente e il resto dopo la divisione.

La modifica dell'algoritmo per sfruttare questa procedura più efficiente può essere guidata direttamente dalla necessità di conservare l'invarianza dell'asserzione $a * b + c = x * y$. L'istruzione di assegnazione $b := b - 1$ nell'algoritmo originale deve essere rimpiazzata da $b :=$ b DIV 10; per conservare l'invarianza, a va quindi moltiplicato per 10 e pertanto l'istruzione $a := a * 10$ va aggiunta al programma. Se b non è un multiplo di 10, il resto (b MOD 10) viene sottratto da b e, sempre per conservare l'invarianza, $a * (b$ MOD 10) va sommato a c.

Dato che i calcolatori utilizzano internamente il sistema binario, è vantaggioso usare un fattore 2 al posto del 10, conveniente nel sistema decimale: si potrebbe semplicemente sostituire 10 con 2, ma sono possibili ulteriori raffinamenti; l'algoritmo risultante è utilizzato in tutti i calcolatori. Il guadagno di efficienza è sostanziale: il numero di iterazioni del ciclo si riduce da y al logaritmo di y in base 2; senza questo miglioramento i calcolatori impiegherebbero la maggior parte del tempo nell'esecuzione di moltiplicazioni.

Un secondo esempio è preso dal campo degli algoritmi non numerici. Dato un testo memorizzato come sequenza di caratteri, si chiede di trovare al suo interno la prima occorrenza di una parola, che può essere definita come qualsiasi sequenza di caratteri non più lunga del testo. Algoritmi costruiti su questo modello sono importanti in molte aree della scienza dei calcolatori e, in particolare, nei programmi per l'elaborazione di testi.

Il primo passo nella costruzione dell'algoritmo è specificare l'esatto risultato richiesto; nell'illustrazione della pagina a fronte questo viene fatto in una notazione formale (il calcolo dei predicati); ne darò qui una descrizione informale. Le due variabili, il testo e la parola, possono essere dichiarate come vettori di caratteri, in modo che ogni carattere desiderato possa essere ottenuto fornendo un valore come indice. Assumiamo che il testo sia un vettore di m caratteri e la parola sia un vettore di n caratteri, dove n è inferiore o uguale a m. (Nella maggior parte dei casi n è molto minore di m.)

Quale condizione è sicuramente soddisfatta quando è stata trovata un'identica sequenza di caratteri? La risposta può essere definita in termini delle variabili indice i e j che specificano rispettivamente posizioni nel vettore contenente la parola e nel vettore contenente il testo. La parola è trovata se per ogni valore di i da 0 a $n - 1$ (vale a dire per tutto l'intervallo di valori ammessi per l'indice) il carattere designato nel vettore contenente la parola è lo stesso del carattere nel vettore contenente il testo specificato dall'indice $j + i$. Il valore di j che soddisfa queste condizioni punta al primo carattere della sequenza corrispondente alla parola e può servire come risultato prodotto dall'algoritmo di ricerca.

La definizione del problema richiede la prima sequenza corrispondente. Bisogna quindi aggiungere all'algoritmo un'altra condizione: per tutti i valori dell'indice j nel vettore contenente il testo inferiori al valore al quale ha inizio la sequenza trovata, il vettore contenente la parola e il vettore contenente il testo debbono differire in almeno un carattere. Il risultato è valido solo se entrambe le condizioni sono soddisfatte.

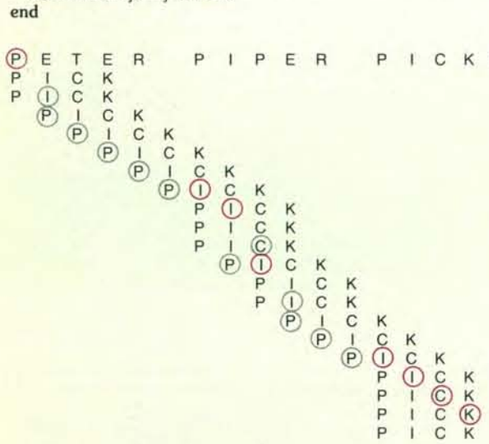
È necessario considerare inoltre un'altra possibilità: la parola può non essere affatto presente nel testo; trascurare casi del genere è causa frequente di malfunzionamento dei programmi. Si possono correggere le condizioni specificando che se la parola non viene trovata il valore di j risultante deve essere superiore al massimo possibile valore di inizio di una sequenza corrispondente, vale a dire $m - n$.

Le tre condizioni sopra definite - che i caratteri nella parola e nel testo corrispondano per tutti i valori di i e $j + i$, che non esistano intervalli di corrispondenza per valori inferiori di j e che j sia inferiore a $m - n$ - rappresentano asserzioni utili nel verificare la correttezza di un algoritmo. Di più, sono anche utili come scheletro per la costruzione dell'algoritmo stesso. Il metodo più ovvio consiste nello scandire ripetutamente il testo. Si costruisce un ciclo *while* con la prima e la terza condizione come guardie e con la seconda condizione come invariante di ciclo. Il valore iniziale di j viene posto a 0, in modo che la ricerca parta dall'inizio del testo. A ogni esecuzione del ciclo le condizioni di guardia vengono esaminate e se la parola è stata trovata o se j è maggiore di $m - n$, il programma esce dal ciclo, altrimenti j viene incrementato di 1 e il ciclo viene ripetuto.

Ciò che resta da specificare è come venga scoperta l'occorrenza della parola nel testo, cioè come i caratteri del testo vengano confrontati con quelli della parola nell'intervallo di indici da $i = 0$ a $i = n - 1$. La risposta è un ciclo entro il ciclo principale: per ogni valore assegnato a j il ciclo interno passa attraverso l'intero intervallo di valori di i, confrontando gli n caratteri uno alla volta. Alla prima discrepanza il ciclo interno viene interrotto. Il valore di i all'uscita dal ciclo indica se un'occorrenza della parola è stata trovata: se i è inferiore a n il confronto è terminato prematuramente a causa di un carattere differente.

L'algoritmo di ricerca in un testo del tipo illustrato è semplice, ma relativamente inefficiente. In sostanza la parola e il testo vengono sovrapposti, partendo dall'inizio di entrambe, e confrontati carattere per carattere; se viene trovato un carattere differente la parola viene spostata di un carattere a destra e il confronto viene ripetuto. Questo processo continua finché la parola viene trovata nel testo oppure la parola

```
testo: array[0..M–1] of character
j =          0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
testo [j] = P E T E R   P I P E R   P I C K E D   A   P E C K

parola: array[0..N–1] of character
i =          0 1 2 3
parola [i] = P I C K

i := 0; j := 0;
while (i < N) and (j < M–N) do
  i := 0;
  while (i < N) and parola[i] = testo[j+i] do i := i+1 end;
  if i< N then j := j +1 end
end
```



P: ∀i: (0≤i<N): parola [i] = testo [j+i]

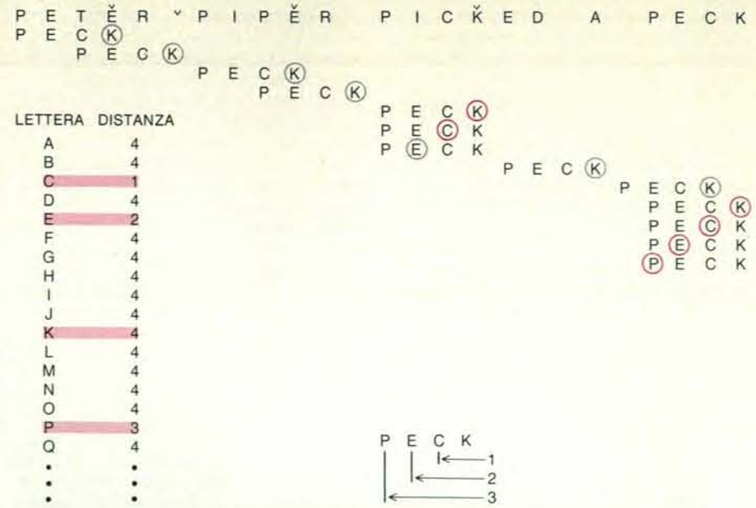
Q: ∀k: (0 ≤k<j): ∃i: (0≤i<N): parola [i] ≠ testo [k+i]

R: P∧Q∧ (j≥M−N ∨ i = N)

∀= per tutti ∨ = o

∃= esiste un ∧ = e

La ricerca di una parola in un testo viene effettuata con una serie di confronti lettera per lettera. Sia il testo sia la parola sono dichiarati vettori di caratteri; in questo caso il testo ha 25 lettere e la parola quattro. La prima lettera della parola viene confrontata con la prima lettera del testo; in questo caso corrispondono *(evento indicato dal cerchio in colore)* e quindi si passa a confrontare la seconda lettera: stavolta differiscono *(evento indicato dal cerchio in grigio);* la parola viene quindi spostata di una posizione a destra e il confronto delle coppie di lettere ricomincia da capo dall'inizio della parola. Solo quando tutte le lettere corrispondono la parola è stata trovata. Le condizioni che l'algoritmo deve soddisfare sono specificate dalle tre proposizioni del calcolo dei predicati mostrate in basso nell'illustrazione. La prima asserzione *(P)* stabilisce che quando la parola è allineata alla posizione *j* nel vettore del testo, per tutti i valori dell'indice *i* del vettore della parola, il carattere *parola* [*i*] è lo stesso del carattere *testo* [*j* + *i*]. La seconda proposizione *(Q)* stabilisce che non vi è alcun'altra sequenza corrispondente per valori inferiori di *j;* pertanto deve venir trovata la prima occorrenza della parola nel testo. La terza proposizione *(R)*, che deve valere alla fine della ricerca, stabilisce che sia *P* sia *Q* siano soddisfatte e o *i* avrà il valore *N* (indicando che è stata trovata un'occorrenza in posizione *j*) o *j* avrà un valore maggiore di quello di ogni possibile sequenza corrispondente (indicando che la parola non è presente nel testo).


P E T Ě R ˇ P I P Ě R P I C Ǩ E D A P E C K
P E C K
P E C K

viene spostata fino alla fine del testo. Se la parola non esiste nel testo, sono necessari almeno $m - n$ confronti e il numero è generalmente molto maggiore.

Per un compito fondamentale come la ricerca in un testo può sembrare improbabile che un metodo significativamente migliore non sia stato scoperto se non dopo circa 30 anni di ricerche nella scienza dei calcolatori; fu solo nel 1976 che Robert S. Boyer e J. Strother Moore II, ora all'Università del Texas ad Austin, trovarono un metodo più veloce. La loro idea permette incrementi di j maggiori di 1 nel ciclo principale del programma. Il confronto della parola con una parte del testo inizia alla fine della parola e procede verso l'inizio. Se una lettera nella parola non è uguale alla corrispondente lettera nel testo, la parola viene spostata in avanti in modo da allineare la successiva lettera uguale a quella nella posizione in esame nel testo, che chiamerò posizione chiave. Se nessuna lettera nella parola è uguale a quella nella posizione chiave, la parola viene spostata in avanti in modo che la sua ultima lettera sia una posizione oltre quella chiave.

Si pone immediatamente il problema di come venga trovata la successiva lettera uguale a quella nella posizione chiave: se questo deve essere fatto confrontando i caratteri uno alla volta, non si guadagna nulla. C'è però un altro modo: il programma può servirsi di una tabella che elenchi la distanza dalla fine della parola all'ultima occorrenza di ogni lettera nella parola. Naturalmente bisogna investire tempo nella compilazione della tabella, ma questo lavoro va svolto una sola volta; se il testo è abbastanza lungo, ne vale la pena.

L'algoritmo di Boyer e Moore è più veloce, ma si può avere fiducia nella sua correttezza? In particolare come si può essere sicuri nello spostare la parola di parecchi posti verso destra senza effettuare confronti che nessun allineamento utile venga saltato? Un'argomentazione informale è che per trovare la parola debbano essere uguali tutte le coppie di caratteri e gli allineamenti saltati differiscono necessariamente in almeno una posizione, quella chiave.

Correttezza ed efficienza sono le principali preoccupazioni dei programmatori. Ho dimostrato come metodi analitici possano e debbano venir usati per stabilire la correttezza, perché un controllo esaustivo empirico richiederebbe troppo tempo anche per problemi semplici. Esattamente per le stesse ragioni l'efficienza e le prestazioni non possono venir misurate empiricamente; lo strumento per la loro analisi è il calcolo delle probabilità.

Supponiamo che venga dato un vettore di n numeri e venga chiesto di cercarne il massimo. Il metodo ovvio consiste nello scandire sequenzialmente il vettore, confrontando ogni elemento con il maggiore trovato fino a quel punto. Si potrebbe dichiarare una variabile di nome *max* e prendere come suo valore iniziale quello del primo elemento; in un ciclo *while* ogni elemento seguente viene confrontato con *max* e, se è maggiore, a *max* viene assegnato il valore dell'elemento.

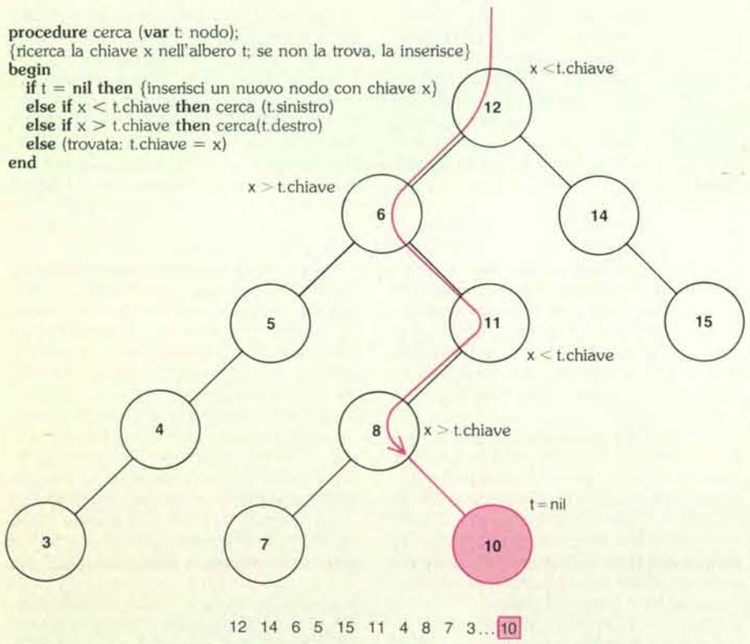
È evidente che il ciclo deve essere ripetuto n volte, il che impone un limite inferiore al tempo di esecuzione della procedura. Con che frequenza viene eseguita l'istruzione di assegnazione? Se il primo elemento è il massimo, l'assegnazione viene eseguita solo una volta; d'altro canto se la sequenza è ascendente, a *max* viene assegnato un nuovo valore n volte. Pertanto 1 e n sono i valori estremi, ma qual è la media? Non è possibile rispondere alla domanda con un esperimento; esistono $n!$ possibili ordinamenti dei numeri, un valore troppo alto, anche per piccoli valori di n. (Con un vettore di soli 16 elementi e un calcolatore che esamini un milione di ordinamenti al secondo, un'analisi esaustiva impiegherebbe oltre mezzo anno.)

Il metodo analitico per determinare la media è piuttosto semplice. L'assegnazione iniziale viene sempre eseguita e pertanto il conto delle esecuzioni comincia da 1; assumendo che tutte le permutazioni siano equiprobabili, la probabilità che il secondo elemento sia maggiore del primo è 1/2; la probabilità che il terzo elemento sia maggiore dei primi due è 1/3. L'analisi prosegue allo stesso modo e quindi il numero medio di assegnazioni è uguale alla somma $1 + 1/2 + 1/3 + ... + 1/n$, che è nota come serie armonica. Leonhard Euler, matematico svizzero del XVIII secolo, dimostrò che la somma della serie è approssimativamente uguale al logaritmo naturale di n più una costante, oggi chiamata costante di Eulero, che vale circa 0,577. Il logaritmo di n cresce molto più lentamente di n; quindi il tempo impiegato nell'assegnare valori a *max* è trascurabile rispetto al tempo impiegato per i confronti e per incrementare l'indice del vettore. È pertanto ragionevole dire che il lavoro richiesto per trovare il massimo tra n numeri è proporzionale a n.

La maggior parte dei problemi pratici non cede così facilmente all'analisi e l'analisi di algoritmi è un attivo campo di ricerca. In molti casi è sufficiente conoscere come il tempo di esecuzione vari in funzione di una qualche misura delle dimensioni del problema. Per esempio il tempo potrebbe crescere proporzionalmente alla dimensione, o al quadrato della dimensione, o potrebbe crescere esponenzialmente; la crescita esponenziale rende l'algoritmo quasi inutile. Lo studio di questi problemi si chiama «analisi di complessità».

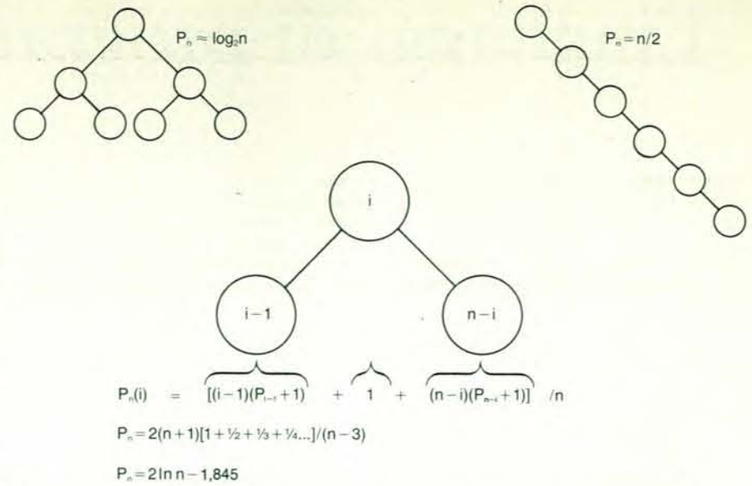
I metodi dell'analisi di complessità possono essere illustrati da un esempio: la costruzione di un albero binario, una struttura di dati adottata spesso quando è importante una rapida ricerca delle informazioni. L'albero ha due proprietà di rilievo; ogni nodo può avere al massimo due sottonodi e le chiavi che identificano i nodi sono ordinate in modo tale che in ogni nodo la chiave minore sta nel sottoalbero di sinistra. La ricerca di una particolare chiave può essere molto efficiente: un confronto a ogni nodo indica se seguire il ramo destro o il sinistro, cosicché il numero di possibilità rimanenti si dimezza a ogni nodo.

```
procedure cerca (var t: nodo);
{ricerca la chiave x nell'albero t; se non la trova, la inserisce}
begin
  if t = nil then {inserisci un nuovo nodo con chiave x}
  else if x < t.chiave then cerca (t.sinistro)
  else if x > t.chiave then cerca(t.destro)
  else (trovata: t.chiave = x)
end
```



Un algoritmo per l'inserimento in ordine casuale può essere codificato con un'unica procedura di gestione di alberi binari che può sia inserire nuove informazioni sia ricercarle; la procedura è ricorsiva e viene applicata esattamente allo stesso modo a ogni nodo. Dato un valore x della chiave da cercare, l'algoritmo lo confronta con la chiave del nodo in esame; se x è inferiore alla chiave contenuta nel nodo, la ricerca prosegue lungo il sottoalbero sinistro; se è maggiore si esamina il sottoalbero destro. Se il valore di x è uguale a quello della chiave, il nodo desiderato è stato trovato. Se il puntatore al nodo è *nil*, il nodo non esiste e se ne crea uno nuovo in quella posizione dell'albero. Nell'esempio mostrato nell'illustrazione viene ricercato un nodo con chiave $x = 10$.

L'efficienza raggiunge il massimo quando l'albero è perfettamente bilanciato, cioè quando ogni nodo ha esattamente due sottonodi. Il cammino medio di ricerca - il numero medio di confronti previsto - è in tal caso uguale al logaritmo in base due del numero dei nodi. Nel caso peggiore, quando l'albero degenera in una semplice lista con un solo sottonodo concatenato a ogni nodo, il cammino medio di ricerca è la metà del numero dei nodi più uno. Sembra quindi che si dovrebbe far sì che l'albero si mantenga bilanciato mentre cresce. L'operazione stessa di bilanciamento richiede però uno sforzo considerevole e diviene lecito chiedersi se tale lavoro aggiuntivo verrà compensato dal tempo risparmiato durante le ricerche. Ma la risposta è generalmente no.



Risulta che il bilanciamento di un albero binario offre solo un modesto miglioramento di efficienza nel caso medio. Un albero perfettamente bilanciato con n nodi *(in alto a sinistra)* ha una lunghezza media di cammino proporzionale al logaritmo di n; nel caso peggiore, quello di un albero degenerato in una lista *(in alto a destra)*, la lunghezza media è $(n + 1)/2$. Per un albero costruito per inserimenti in ordine casuale la lunghezza media deve essere determinata con un'analisi probabilistica. Se la radice dell'albero ha chiave i, il sottoalbero sinistro deve avere $i - 1$ nodi e quello destro $n - i$. La lunghezza media totale del cammino è uguale alla somma pesata delle lunghezze dei due sottoalberi più uno per la radice; l'analisi può essere ripetuta per ogni sottoalbero, finché alla fine vengono raggiunte le foglie, dove ogni sottoalbero ha una lunghezza di cammino di 0 o 1. La lunghezza media del cammino nell'albero creato con inserimenti casuali è una funzione logaritmica di n, maggiore di circa il 38 per cento della lunghezza nel caso dell'albero bilanciato.

Supponiamo che n valori di chiavi siano lette in ordine casuale e inserite in un albero inizialmente vuoto. L'ordinamento da sinistra a destra delle chiavi viene mantenuto man mano che le stesse vengono inserite, ma non viene effettuato alcun tentativo di bilanciare l'albero. Un'unica procedura può essere progettata sia per inserire chiavi nuove sia per cercarne una già presente; la procedura cerca semplicemente il punto nell'albero dove la chiave dovrebbe trovarsi e la inserisce se non la trova. Un algoritmo per questo scopo è mostrato nell'illustrazione della pagina a fronte. Si tratta di una procedura ricorsiva, cioè una procedura che richiama se stessa. A ogni nodo l'algoritmo intraprende una delle quattro possibili azioni: se il valore della chiave nel nodo è *nil*, la nuova chiave viene inserita; se il valore della chiave del nodo è uguale a quello della nuova chiave, viene riferito il successo della ricerca; se la nuova chiave è inferiore a quella trovata, la procedura chiama una copia di se stessa per cercare nel sottonodo di sinistra; se la nuova chiave è maggiore, la ricerca ricorsiva viene intrapresa a destra.

Qual è la lunghezza del cammino medio in un albero costruito con un simile inserimento casuale? Di nuovo misurazioni empiriche non possono essere prese in considerazione, poiché esistono $n!$ possibili sequenze di inserimento, ma si può effettuare una stima probabilistica. Supponiamo che le chiavi siano gli interi da 1 a n e che tutte le loro permutazioni siano equiprobabili; una chiave, i, deve essere la prima ad arrivare e diviene pertanto la radice dell'albero. Quando le rimanenti chiavi saranno state inserite, esisteranno $i - 1$ nodi a sinistra della radice e $n - i$ nodi a destra. Se i è il punto centrale dell'intervallo, l'albero è perfettamente bilanciato al livello più alto; se i è uguale a 1 o a n, i primi rami dell'albero sono completamente sbilanciati.

L'idea centrale nell'analisi è che esattamente lo stesso ragionamento può essere applicato ricorsivamente a ogni sottoalbero. Se la seconda chiave ad arrivare è j ed è inferiore a i, allora quando l'albero viene riempito, esisteranno $j - 1$ nodi nel ramo a sinistra di j e $i - j$ nodi nel ramo a destra. Da questa descrizione ricorsiva dell'albero la lunghezza media del cammino può essere calcolata da un'altra procedura ricorsiva. Alla radice la lunghezza del cammino è uguale a 1 (per contare il nodo della radice stessa) più la lunghezza di un sottoalbero con $i - 1$ nodi, più la lunghezza di un sottoalbero con $n - i$ nodi. Queste lunghezze non sono ancora note, ma si possono calcolare applicando la stessa procedura al successivo livello dell'albero. Prima o poi viene raggiunta la fine di ogni ramo, dove ogni nodo ha una lunghezza di cammino di 0 o 1.

La definizione ricorsiva di lunghezza di cammino deve essere mediata per tutti i possibili valori di i da 1 a n. Il risultato, mostrato nell'illustrazione di questa pagina, è un'espressione che include ancora la serie armonica. La lunghezza media del cammino di un albero costruito senza preoccuparsi di bilanciarlo è una funzione logaritmica del numero di nodi e differisce da quella del caso ottimale per un fattore costante. Il caso medio è molto più vicino a quello ottimale che al peggiore: è più lungo del 38 per cento.

I programmi qui discussi non sono banali, ma sono molto corti; in applicazioni pratiche i programmi tendono a essere lunghi e intricati e sembrano crescere tanto velocemente quanto la disponibilità di memoria dei calcolatori sui quali vengono eseguiti. È possibile applicare a programmi del genere i metodi di analisi che ho delineato? Sono profondamente convinto che si debba, poiché sistemi complessi richiedono un ragionamento esatto in maniera anche più impellente di quelli semplici.

La maggior parte dei grandi sistemi software si affidano a pochi algoritmi «profondi»; per lo più sono costruiti mediante algoritmi fondamentali, come quelli di moltiplicazione e ricerca, che compaiono in molte variazioni e combinazioni. Le strutture di dati, invece, tendono a essere incredibilmente complesse. Come conseguenza la scelta della corretta rappresentazione per i dati è spesso la chiave per programmare con successo e può avere un'influenza maggiore sulle prestazioni del programma dei dettagli dell'algoritmo impiegato.

È improbabile che esisterà mai una teoria generale per la scelta delle strutture di dati. Il meglio che possa essere fatto è comprendere gli elementi fondamentali e le strutture che con essi è possibile costruire. L'abilità di applicare questa conoscenza nella costruzione di sistemi di grandi dimensioni è principalmente una questione di abilità ingegneristica e di esperienza. Nell'acquisire questa abilità il programmatore deve combattere costantemente la complessità, rifiutarsi di far ricorso a metodi non pienamente compresi e non abbandonare mai la ricerca di soluzioni più semplici e più eleganti. In questo sforzo nessun moderno strumento di ingegneria del software può sostituire la capacità di un ragionamento, preciso e costruttivo, del programmatore.

Linguaggi di programmazione

Servono per specificare i procedimenti di elaborazione, ma sono molto più di semplici notazioni: trasformano il calcolatore in una «macchina virtuale» le cui caratteristiche e capacità sono determinate dal software

di Lawrence G. Tesler

Un linguaggio di programmazione è più di una notazione per fornire istruzioni a un calcolatore: un linguaggio e il software che lo può «capire» possono ricostruire completamente il calcolatore, trasformandolo in una macchina con un carattere completamente diverso. I componenti fisici di un tipico calcolatore sono registri, celle di memoria, addizionatori e così via, e quando un programmatore scrive nel linguaggio nativo del calcolatore questi sono i dispositivi che deve aver presenti. Un nuovo linguaggio porta con sé un nuovo modello della macchina. Anche se l'hardware è immutato, il programmatore può pensare in termini di variabili anziché di celle di memoria, di archivi di dati anziché di canali di ingresso e di uscita, e di formule algebriche anziché di registri e addizionatori. Taluni linguaggi addirittura danno al calcolatore una personalità multipla: la macchina diventa un insieme di entità indipendenti che eseguono ciascuna i propri calcoli e inviano messaggi l'una all'altra.

Esistono centinaia, se non addirittura migliaia, di linguaggi di programmazione e di dialetti di linguaggi di programmazione. Le lingue naturali della comunicazione umana probabilmente sono ancora più numerose, ma sotto certi punti di vista fra i linguaggi di programmazione le differenze sono più marcate. Ogni linguaggio ha la sua grammatica e la sua sintassi, il suo modo di esprimere idee. In linea di principio quasi tutti i compiti computazionali potrebbero essere eseguiti con qualunque linguaggio, ma i programmi risulterebbero molto diversi; inoltre, scrivere un programma per un dato compito sarebbe molto più facile con certi linguaggi che non con altri. Verranno qui descritti alcuni fra i linguaggi di programmazione di uso comune e si cercherà di dare un'idea dei loro elementi comuni e delle caratteristiche che li differenziano.

L'illustrazione a pagina 40 mostra vari stadi nello sviluppo di un programmino in Logo, un linguaggio di programmazione formulato verso la fine degli anni sessanta da Seymour Papert e collaboratori al Massachusetts Institute of Technology (MIT). Una caratteristica interessante del Logo è il fatto che permetta di controllare una «tartaruga», un piccolo robot che può muoversi in avanti e all'indietro, può ruotare in un punto e può alzare o abbassare una penna che, se abbassata, lascia una traccia del percorso della tartaruga stessa. In genere la tartaruga non è un dispositivo meccanico, ma è simulata sullo schermo del calcolatore.

La versione iniziale del programma è costituita esclusivamente da comandi alla tartaruga. Dapprima si abbassa la penna, poi si ripetono cinque volte i comandi *avanti 50* e *destra 144*, infine si solleva la penna. Quando la tartaruga segue le istruzioni, disegna una stella a cinque punte. Il comando *avanti 50* fa sì che disegni un segmento lungo 50 unità; *destra 144* specifica una rotazione in senso orario di 144 gradi, cioè il cambiamento di direzione in corrispondenza di ciascun vertice di una stella a cinque punte.

Se scrivere un elenco di comandi che debbono essere eseguiti in successione fosse l'unico metodo per comunicare le proprie intenzioni a un calcolatore, la creazione di un programma complesso sarebbe praticamente impossibile. In effetti il Logo e altri linguaggi di programmazione mettono a disposizione numerosi strumenti per semplificare e generalizzare le istruzioni. In questo caso la parte del programma che più ha bisogno di un perfezionamento è la ripetizione, per cinque volte, degli enunciati *avanti* e *destra*. Non appena può, un programmatore evita di scrivere qualunque cosa più di una volta, e non solo perché scrivere sia faticoso. Se il programma potesse essere condensato, occuperebbe un minore spazio di memoria; inoltre, le ripetizioni aumentano la probabilità di un errore tipografico, in particolare durante la revisione del programma. La ripetizione può essere eliminata sostituendo i cinque comandi di movimento della tartaruga con l'enunciato *ripeti 5 [avanti 50 destra 144]*.

Supponiamo ora che il programmatore voglia disegnare una stella a nove punte con i lati di 80 unità. L'obiettivo può essere raggiunto con un enunciato come *ripeti 9 [avanti 80 destra 160]*, ma è chiaro che si sta duplicando la stessa struttura di programma, con differenze solo di dettaglio. Una soluzione migliore sta nel definire una procedura più generale in cui il numero delle punte e la lunghezza di un lato sono dati come grandezze variabili. In Logo una definizione di procedura è introdotta dalla parola *per*. Così il sintagma *per stella* indica che le istruzioni che seguono vanno memorizzate come il metodo per disegnare una stella. Dopodiché *stella* diventa un nuovo comando del linguaggio, che può essere inserito in un programma esattamente come è stato fatto per i comandi, presenti dall'origine nel linguaggio, *avanti* e *destra*.

Le variabili nella procedura *stella* sono del tipo denominato parametri, che vengono «passate» alla procedura nel momento in cui viene chiamata. In Logo il nome di un parametro è preceduto da un segno di due punti. Pertanto la procedura sarebbe definita con un sintagma come *per stella :dimensione :punte;* e battendo *stella 80 9* si assegnerebbe valore 80 al parametro *dimensione* e valore 9 a *punte*, generando così una stella a 9 punte con lati di 80 unità.

Si potrebbe apportare un ulteriore miglioramento alla procedura *stella*. In Logo una procedura definita può essere chiamata non solo dal programmatore, ma anche da un'altra procedura. La potenza espressiva aumenta notevolmente, ma aumenta anche il rischio che a una procedura vengano forniti parametri non adeguati. Per esempio, persi nei meandri di un programma complesso, potrebbe capitarci di non renderci conto che alla tartaruga venga chiesto di disegnare una stella con una o due punte solamente. Il problema può essere evitato aggiungendo al programma la clausola *se punte* > 2. La clausola condizionale funge da «guardia» che consente alla tartaruga di disegnare solo se il numero di punte specificato è maggiore di due.

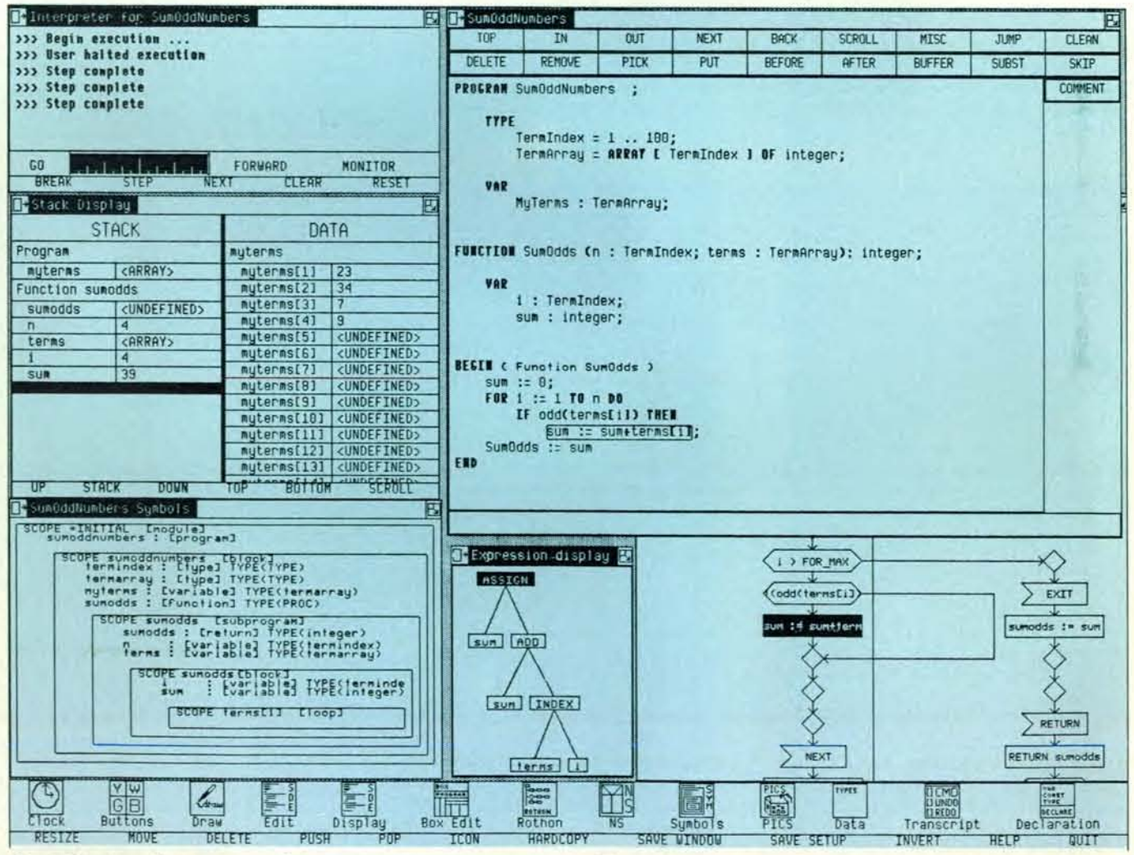
Dall'esempio si può vedere che il Logo ha almeno una somiglianza superficiale con una lingua naturale come l'italiano. (Il caso del Logo è in effetti fortunato perché esistono versioni italiane del linguaggio, alle quali abbiamo fatto riferimento nel testo precedente: per la maggior parte dei linguaggi di programmazione, in realtà, la somiglianza è al più con la lingua inglese.) Possiede un vocabolario di parole, numeri e altri simboli che possono essere combinati in successione per realizzare costruzioni di maggiori dimensioni, analoghe alle frasi. Alcuni dei simboli sono «parole chiave» o «riservate» con un significato prestabilito: altri sono definiti dal programmatore. Alcuni elementi fungono da verbi, altri hanno funzioni analoghe a quelle dei nomi, dei modificatori o dei segni di interpunzione. Le regole che governano le possibili combinazioni di simboli costituiscono una grammatica.

Solitamente, le frasi di un linguaggio di programmazione vengono classificate in due categorie: dichiarazioni e istruzioni. Una dichiarazione definisce che cosa sia una certa cosa, che cosa significhi e come sia strutturata. Nel programma in Logo, *per stella :dimensione :punte* è una dichiarazione che definisce *stella* come il nome di una procedura e definisce *dimensione* e *punte* come variabili che fungono da parametri per la procedura *stella*. Un'istruzione, invece, in genere descrive una parte di un algoritmo: specifica qualche azione che debba essere intrapresa. Nella maggior parte dei casi un'istruzione ha la forma di una frase imperativa: comincia con un verbo, seguito da un complemento oggetto o da un modificatore. Nell'istruzione *ripeti*, lo stesso *ripeti* è un verbo, il numero che lo segue funge da avverbio e tutto ciò che si trova fra parentesi è il complemento oggetto del verbo.

Il vocabolario e la sintassi nella procedura *stella* sono tipici del Logo, ma il meccanismo che controlla il flusso dell'esecuzione nel corso della procedura si può trovare nella maggioranza dei linguaggi di programmazione. In assenza di un'istruzione di controllo esplicita, l'esecuzione è sequenziale. Se si immagina il calcolatore come una persona che legge il programma, lo leggerà dalla prima riga in alto verso l'ultima in basso e quindi, a meno di modifiche al flusso di controllo, ogni istruzione sarà eseguita esattamente una volta.

Un elemento che, all'interno di un programma, modifica il flusso di esecuzione è l'istruzione *ripeti*, un esempio di costrutto iterativo. Quando incontra *ripeti n* seguito da un gruppo di istruzioni racchiuse fra parentesi, il calcolatore legge ed esegue *n* volte le istruzioni contenute fra parentesi. Un altro modo per controllare il procedere del calcolatore lungo un programma è l'esecuzione condizionale, che in molti linguaggi di programmazione è realizzata mediante l'istruzione *if* (è così anche nella versione inglese del Logo; nella versione italiana si trova *se*). Le istruzioni controllate da un *if* vengono eseguite solo se è soddisfatta qualche condizione specificata.

Esistono molte variazioni sul tema base dell'esecuzione iterativa e condizionale. L'istruzione *ripeti* è utile solo se si sa in anticipo quante volte il ciclo debba essere eseguito. Altri costrutti permettono il controllo della conclusione del ciclo me-



Questa istantanea di un programma in esecuzione è data da un sistema di sviluppo di programmi chiamato PECAN. Gran parte del testo sorgente, cioè del programma originale, è visualizzata nella grande finestra a destra in alto: è un programma in Pascal per sommare i termini dispari in un vettore di interi. I comandi che controllano l'esecuzione del programma sono dati nella finestra a sinistra in alto. L'esecuzione è stata fermata all'istruzione che permette l'effettivo calcolo della somma; l'istruzione è racchiusa in un riquadro nella finestra di visualizzazione del testo sorgente. Sotto il testo sorgente è visibile parte del diagramma di flusso del programma e, immediatamente a sinistra, vi è un albero binario che mostra la struttura logica dell'istruzione di assegnamento. I riquadri annidati a sinistra in basso indicano il campo di validità dei simboli nel programma. La finestra di visualizzazione della catasta o pila (*stack*) fornisce un'immagine delle strutture di dati del programma. Il PECAN è stato sviluppato da Steven P. Reiss che lavora alla Brown University e al quale va anche attribuita questa illustrazione.

diante eventi interni al ciclo stesso: sostanzialmente, viene incorporata all'interno del ciclo un'espressione condizionale. In numerosi linguaggi, un'istruzione che cominci con la parola chiave *while* (letteralmente, «mentre») è ripetuta finché resta vera una determinata condizione, formulata esplicitamente. Un altro modo per modificare l'andamento sequenziale del flusso esecutivo è il «salto incondizionato», realizzato con l'istruzione *goto*, che fa passare l'esecuzione a un nuovo punto del programma. Negli ultimi anni l'istruzione *goto* non ha goduto di molto favore presso i teorici della programmazione, perché i programmi che contengono molti salti incondizionati sono difficili da seguire (per una persona che li legga, non per il calcolatore).

Lo stesso concetto di definizione di una procedura è un elemento vitale della programmazione. Costituisce infatti il meccanismo principe dell'astrazione, il processo mediante il quale i casi specifici (una stella a cinque punte con il lato di 50 unità) vengono trasformati in concetti generali (una stella, con un numero qualunque di punte e lati di lunghezza qualunque). Una procedura è definita una sola volta ed è memorizzata una sola volta, ma può essere richiamata in molti punti diversi di un programma: in questo modo il frutto della fatica mentale può essere utilizzato ripetutamente. Ogni volta che si chiama una procedura, l'esecuzione è trasferita all'area di memoria in cui la procedura è immagazzinata; quando la procedura è stata completata, il controllo ritorna all'istruzione immediatamente successiva a quella in cui era avvenuta la chiamata della procedura. Un tipo particolare di procedura, una funzione, fornisce al programma chiamante un valore: per esempio, la funzione tangente riceve in ingresso un angolo come parametro, e fornisce un valore, che è la tangente di quell'angolo.

Fra le centinaia o migliaia di linguaggi di programmazione, solo una decina o poco più è largamente usata. Le illustrazioni da pagina 42 a pagina 44 mostrano lo stesso problema risolto mediante programmi in sei linguaggi: BASIC, Pascal, COBOL, Forth, APL e Lisp. Ho scelto questi linguaggi perché sono ben conosciuti, perché esistono molti programmatori che li sanno usare e perché, inoltre, esemplificano bene la varietà di modi in cui si può esprimere una medesima idea. In ciascun caso ho tentato di scrivere in uno stile che fosse naturale o almeno comodo per un programmatore abituato a quel linguaggio.

Il problema negli esempi non è di particolare interesse intrinseco. L'ho scelto perché è possibile programmarne facilmente una soluzione in tutti i linguaggi e perché dimostra i meccanismi essenziali per definire variabili e procedure e per controllare l'esecuzione iterativa e condizionale. Il problema è quello di trovare la somma dei numeri dispari che fanno parte di un insieme di interi.

Il linguaggio di programmazione BASIC è stato sviluppato nel 1965 da John G. Kemeny e Thomas E. Kurtz del Dartmouth College, in primo luogo come linguaggio per i corsi introduttivi di scienza del calcolatore. Da allora è sceso nella stima del mondo accademico, ma in compenso si è diffuso in altri contesti, e segnatamente nella programmazione dei microcalcolatori. In BASIC ogni riga è identificata da un numero, e il controllo del flusso lungo il programma si basa prevalentemente sul riferimento ai numeri di riga. Il nocciolo del programma esemplificativo è un ciclo in cui si eseguono ripetutamente tutte le istruzioni fra un'istruzione *for* e un'istruzione *next*. Il calcolo effettivo avviene in un'istruzione di assegnazione, che inizia con la parola chiave *let* e attribuisce un nuovo valore a una variabile.

Il Pascal è stato definito intorno al 1970 da Niklaus Wirth della Eidgenossische Technische Hochschule di Zurigo: è un altro linguaggio pensato per l'insegnamento e adattato a molti altri scopi. A differenza del BASIC, richiede che il programmatore dichiari ogni variabile e ne specifichi il tipo; in questo caso le variabili sono interi e vettori di interi. Il riferimento a procedure e funzioni avviene mediante nomi anziché mediante numeri di riga, il che migliora la leggibilità dei programmi. Il Pascal è stato particolarmente importante come progenitore di linguaggi successivi. Per esempio, Wirth ha definito di recente un linguaggio, Modula-2, che usa molti fra i concetti introdotti in Pascal, ma accentua fortemente la costruzione dei programmi come insiemi di moduli indipendenti. Anche l'Ada, un linguaggio sviluppato sotto l'egida del Department of Defense statunitense, si basa ampiamente sul Pascal, anche se è notevolmente più complesso.

Il COBOL è stato creato nel 1960 da un comitato congiunto costituito da costruttori e utenti di calcolatori. Il nome è un acronimo per COmmon Business-Oriented Language (linguaggio comune orientato al mondo degli affari) e il COBOL è da molto tempo il linguaggio principale per l'elaborazione di dati su grande scala nella pubblica amministrazione, negli istituti di credito, nelle compagnie di assicurazioni e in altri settori analoghi. Un programma in COBOL è costituito da quattro sezioni o parti: quella di identificazione, quella dell'ambiente, quella dei dati e quella delle procedure. Nell'illustrazione in basso a pagina 42 compaiono solo la sezione dei dati *(data division)*, in cui sono dichiarate le variabili, e la sezione delle procedure *(procedure division)*, in cui vengono descritti gli algoritmi. Molti linguaggi di programmazione sono modellati sulla notazione matematica o su quella della logica formale; il COBOL invece è modellato sulla sintassi dell'inglese: i programmi risultano così molto leggibili ma spesso molto verbosi.

Il Forth è stato inventato attorno al 1970 da Charles H. Moore, allora al National Radio Astronomy Observatory. L'obiettivo era di creare un linguaggio per il controllo di processo, in particolare per il controllo dei telescopi, ma anche in questo caso l'uso del linguaggio si è esteso ad altri settori. È stato adottato per molte applicazioni sui mini- e microcalcolatori, in parte perché i programmi in Forth in genere occupano poca memoria. A differenza di quello che accade con il COBOL, i programmi in Forth sono di difficile lettura ed estremamente concisi; molte parole chiave sono segni di interpunzione.

Nel Forth il dispositivo fondamentale del calcolatore è la «catasta» o «pila» *(stack)*, un'area di memoria organizzata come una pila di vassoi, dimodoché il primo elemento posto è l'ultimo che può esserne estratto. Nella versione in Forth del programma campione si dà per scontato che la matrice di numeri e la sua dimensione siano in cima alla catasta, al momento della chiamata della funzione; tutti i calcoli sono effettuati sulla catasta e il valore della funzione è restituito in cima ad essa. Non sono definite variabili.

L'APL ha una sintassi ancor più concisa di quella del Forth. Le iniziali stanno per *A Programming Language*, ma nel 1961, quando fu pubblicato il primo libro sull'APL (l'autore era Kenneth E. Iverson della International Business Machines Corporation), il linguaggio era semplicemente una notazione per esprimere problemi di matematica applicata: la sua implementazione (cioè la realizzazione di un sistema effettivamente programmabile mediante quel linguaggio) su un calcolatore venne più tardi. Una caratteristica distintiva dell'APL è che può trattare con la stessa facilità un'intera matrice di numeri e un valore singolo; un comando per la somma di due numeri può essere applicato senza alcuna modificazione a matrici con migliaia di elementi. Il programma esemplificativo in APL somma gli elementi dispari di una matrice in un unico enunciato. Il valore di ciascun numero modulo 2 identifica gli elementi dispari, che quindi vengono estratti dalla matrice e sommati tra loro.

Sotto alcuni aspetti il Lisp è il più semplice dei linguaggi presi in considerazione qui. Ha solo un tipo di istruzione, per la precisione una chiamata di funzione: la sua grande potenza sta nel fatto che il valore prodotto da una funzione può essere un'altra funzione. Il Lisp è stato sviluppato verso la fine degli anni cinquanta da John McCarthy, a quell'epoca al MIT, e da allora è il linguaggio d'elezione per tutti coloro che inseguono l'obiettivo dell'intelligenza artificiale. Il nome deriva da *list processing*, elaborazione di liste: tanto i programmi quanto i dati sono strutturati come liste.

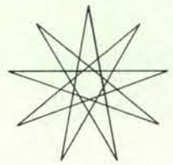
Il programma esemplificativo potrebbe essere scritto in Lisp come un ciclo iterativo, ma un programmatore esperto del linguaggio probabilmente sceglierebbe una tecnica ricorsiva, in cui una procedura chiami se stessa; la procedura chiamata quindi chiama se stessa, e così via. Si deve fornire qualche mezzo di uscita, altrimenti la ricorsione diventa un regresso all'infinito, e per questo solitamente si usa un'istruzione condizionale all'interno della procedura: quando la condizione è soddisfatta, l'esecuzione ritorna al livello di ordine immediatamente superiore.

Nel programma esemplificativo l'insieme dei numeri prende la forma di una lista concatenata. Se la lista è vuota, la funzione produce come risultato zero; se il primo elemento della lista è dispari, viene addizionato alla somma; la funzione richiama comunque se stessa, prendendo come nuovo argomento la lista che rimane dopo l'eliminazione del primo elemento. Alla fine tutti gli elementi saranno stati tolti in questo modo e, a quel punto, tutta la catena di calcoli sarà completata.

```
pennagiu
avanti 50 destra 144
avanti 50 destra 144
avanti 50 destra 144
avanti 50 destra 144
avanti 50 destra 144
pennasu
```

```
pennagiu
ripeti 5 [avanti 50 destra 144]
pennasu
```

```
pennagiu
ripeti 9 [avanti 50 destra 160]
pennasu
```

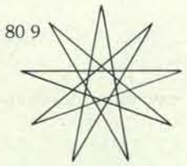


```
per stella :dimensione :punte
  pennagiu
  ripeti :punte [avanti :dimensione destra 720/:punte]
  pennasu
```

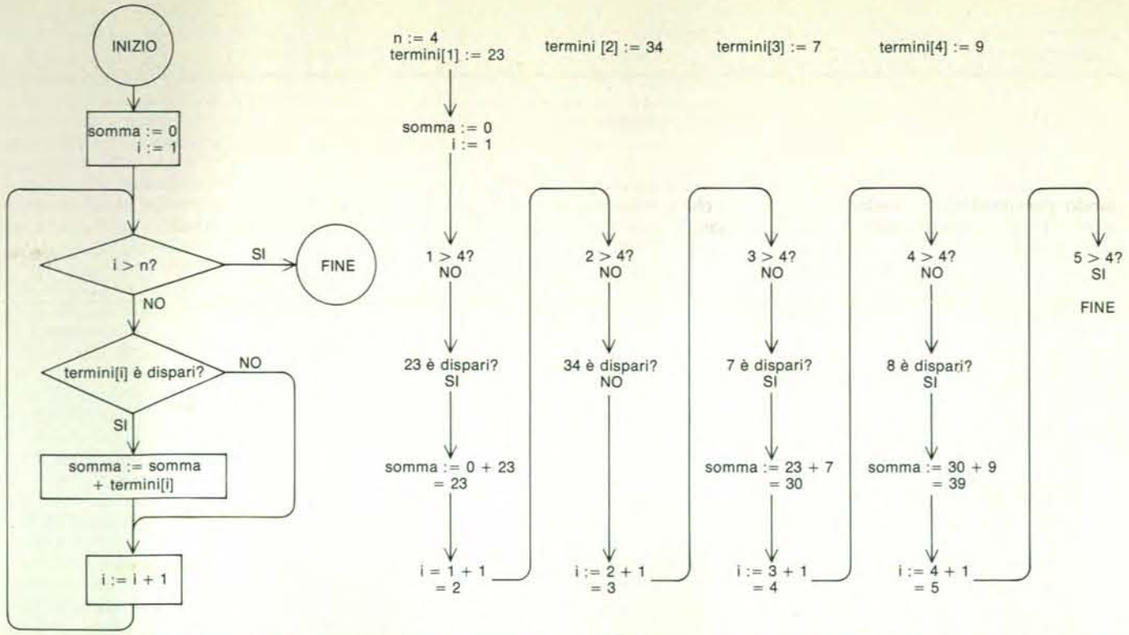
stella 50 5

```
per stella :dimensione :punte
  se :punte > 2
    [pennagiu
    ripeti :punte [avanti :dimensione destra 720/:punte]
    pennasu]
```

stella 80 9



Lo sviluppo di un programma in Logo è articolato in cinque stadi. Il programma dà istruzioni a una «tartaruga», un dispositivo meccanico per il disegno. Nella prima versione, sono date esplicitamente tutte le istruzioni per disegnare una stella a cinque punte. Nella seconda versione, l'enunciato *ripeti* condensa il programma e riduce la probabilità di errore. La terza versione ha la stessa struttura del programma di base, ma disegna una stella più grande con nove punte. Nella quarta versione viene definita una procedura nella quale la lunghezza di un lato e il numero delle punte sono grandezze variabili. Nella versione finale una clausola *se* consente di realizzare la procedura solo se il numero di punte specificato è maggiore di due. Gli enunciati *ripeti* e *se* sono esempi di strutture di controllo importanti praticamente in tutti i linguaggi di programmazione.



Per illustrare le caratteristiche di vari linguaggi di programmazione è stato utilizzato un programma che trova la somma degli elementi dispari in un vettore di *n* interi. L'algoritmo nel diagramma di flusso a sinistra è tradotto direttamente nei programmi in Pascal, BASIC e COBOL di pagina 42. Il suo punto centrale è un ciclo eseguito *n* volte. A ogni passaggio viene esaminato un termine del vettore; se è un numero dispari, si somma al totale corrente. A destra si vede come procede l'algoritmo, con i valori assunti successivamente dalle variabili, quando il calcolo è effettuato per un vettore di quattro numeri. Il simbolo «:=» assegna alla variabile alla sua sinistra il valore calcolato alla sua destra. Un numero fra parentesi quadre, come in *termini* [*I*], è equivalente a un indice sottoscritto: identifica un elemento del vettore *termini*.

```
program SommaNumeriDispari;
type IndiceTermini = 1..100;
     VettoreTermini = array [Indice Termini] of integer;
var  mieiTermini: VettoreTermini;

function SommaDispari(n: IndiceTermini; termini: VettoreTermini): integer;
  var i: Indice Termini
      somma: integer;
  begin
    somma : = 0;
    for i := 1 to n do
      if odd(termini[i]) then
            somma := somma + termini(i);
    SommaDispari := somma;
  end;

begin
  mieiTermini[1] := 23; mieiTermini [2] := 34; mieiTermini [3] := 7; mieiTermini [4] := 9;
  WriteLn(SommaDispari(4, mieiTermini))
end.
```

Questo programma in Pascal per la somma dei numeri dispari in un vettore usa una funzione chiamata *SommaDispari* con due parametri: un intero *n* e un vettore *termini*. La funzione è costituita dalle istruzioni evidenziate dal fondino in colore; il resto del programma costruisce un particolare vettore su cui *SommaDispari* opera. In Pascal ogni variabile deve essere introdotta mediante una dichiarazione che ne precisi il tipo. Alcuni tipi, come *integer*, sono forniti dal linguaggio di programmazione; altri, come *IndiceTermini*, sono definiti dal programmatore. Il ciclo è evidenziato dall'istruzione *for... to... do...* e il condizionale è realizzato da *if...then*.

```
100 DIM T(100)
200 READ N
300 FOR I = 1 TO N
400    READ T(I)
500 NEXT I
600 GOSUB 1100
700 PRINT S
800 GOTO 2000
900 DATA 4
1000 DATA 23, 34, 7, 9
1100 REM S = SOMMA DEGLI ELEMENTI DISPARI NELLA MATRICE T(1..N)
1200 LET S = 0
1300 FOR I = 1 TO N
1400    IF NOT ODD(T(I)) THEN GOTO 1600
1500    LET S = S + T(I)
1600 NEXT I
1700 RETURN
2000 END
```

Il programma in BASIC impiega un sottoprogramma per sommare i termini dispari in un vettore. Il sottoprogramma, indicato dal fondino in colore, non ha nome, e ci si può riferire ad esso solo con il numero di riga; è richiamato dall'istruzione *GOSUB 1100*. Un sottoprogramma in BASIC non ha parametri: i valori sono assegnati a variabili «globali», a cui poi esso può accedere. Una variabile in BASIC non deve essere dichiarata, a meno che abbia indici, come un vettore; in questo esempio la dichiarazione *DIM* (per «dimensione») specifica che il vettore *T* può avere fino a 100 elementi. L'istruzione *FOR... NEXT...* definisce un ciclo e l'istruzione *IF... THEN...* un condizionale.

```
DATA DIVISION.
WORKING STORAGE SECTION.
01 VARIABILI_NUMERICHE USAGE IS COMPUTATIONAL.
   02 TERMINI PICTURE 9999 OCCURS 100 TIMES INDEXED BY I.
   02 N PICTURE 999.
   02 SOMMA PICTURE 999999.
   02 SEMITERMINE PICTURE 9999.
   02 RESTO PICTURE 9.
PROCEDURE DIVISION.
ESEMPIO.
   MOVE 23 TO TERMINI(1).
   MOVE 34 TO TERMINI(2).
   MOVE  7 TO TERMINI(3).
   MOVE  9 TO TERMINI(4).
   MOVE  4 TO N.
   PERFORM SOMMADISPARI.
SOMMA_DISPARI.
   MOVE 0 TO SOMMA.
   PERFORM CONSIDERA_UN_TERMINE VARYING I FROM 1 BY 1 UNTIL I > N.
CONSIDERA_UN_TERMINE.
   DIVIDE 2 INTO TERMINI(I) GIVING SEMITERMINE REMAINDER RESTO.
   IF RESTO IS EQUAL TO 1: ADD TERMINI(I) TO SUM.
```

Il programma in COBOL per il calcolo della somma dei dispari usa una procedura, chiamata *SOMMA_DISPARI*, che ne richiama un'altra: *CONSIDERA_UN_TERMINE*. Una procedura in COBOL non può avere parametri e così prima che *SOMMA_DISPARI* venga richiamata da un'istruzione *PERFORM*, si assegnano valori a *N* e ai primi *N* elementi di *TERMINI*. Le parole chiave *PERFORM... VARYING...* definiscono il ciclo e *IF...* introduce un condizionale. Nella sezione dei dati 01 e 02 designano due livelli in una gerarchia di strutture di dati. *PICTURE* specifica come si debbano memorizzare i valori. È mostrato solo un brano del programma completo.

In generale, un calcolatore non «comprende» il Logo, il BASIC o altri linguaggi a un analogo livello di astrazione. I circuiti del calcolatore riconoscono solamente la forma elettronica dei numeri binari. Un programma memorizzato in una forma eseguibile (quello che viene chiamato «codice macchina») è una successione di numeri binari, alcuni dei quali rappresentano istruzioni per l'unità centrale di elaborazione, altri sono dati e altri ancora indirizzi in memoria.

È possibile scrivere un programma direttamente in codice macchina, ma è un procedimento noioso, e la possibilità di condurre a termine anche un piccolo progetto senza errori è davvero minima. (Il calcolatore non incontra nessuna difficoltà a distinguere 01101001 da 01011001 e a ricordare che cosa significhi ciascuno di questi codici, ma l'occhio e la mente di un programmatore sono un poco a malpartito in questi compiti.) Verso la fine degli anni quaranta e agli inizi del decennio successivo, nel tentativo di alleviare la fatica della stesura di programmi in codice macchina, i programmatori inventarono una notazione chiamata codice assemblatore. Invece di scrivere le cifre binarie di ogni istruzione di macchina, il programmatore scrive una breve parola o un'abbreviazione, come ADD, SUB o MOVE. Analogamente, l'indirizzo di memoria in cui è immagazzinata una variabile è sostituito, in questa notazione, da un nome di variabile. I valori numerici sono espressi in notazione decimale. Le parole che rappresentano le istruzioni sono state scelte in modo da potersi ricordare più facilmente dei valori binari e per questo vengono spesso definite «codici mnemonici».

Inizialmente la traduzione dal codice assemblatore al codice macchina era effettuata a mano. Si tratta comunque di un processo immediato: una tabella registra la corrispondenza fissa fra codici mnemonici e relativi codici binari, e una tabella analoga può essere costruita per i nomi di variabili che appaiono in un programma. Si tratta, chiaramente, di un processo adatto alla meccanizzazione e, infatti, ben presto vennero scritti programmi di traduzione, chiamati assemblatori.

A volte si programma ancora in codice assemblatore, perché si può avere un accesso diretto a tutte le risorse del calcolatore. Un programma in codice assemblatore scritto bene è veloce ed efficiente, e se si deve fare un compromesso fra velocità di esecuzione e lunghezza del programma, il programmatore ha il controllo diretto di una decisione del genere. Gli assemblatori moderni sono raffinati programmi di traduzione. Ciononostante, rimane ancora una corrispondenza biunivoca fra le righe del codice assemblatore e le istruzioni del codice macchina, cosicché i programmi tendono a essere abbastanza lunghi e le possibilità di errore sono enormi. Nella maggior parte dei linguaggi per assemblatore le strutture di controllo disponibili sono primitive. Cosa ancora più importante, il codice assemblatore resta più vicino al linguaggio della macchina che a quello del programmatore.

Gli algoritmi debbono essere espressi in termini di quello che la macchina deve fare, anziché nei termini che potrebbero essere più naturali per il problema volta a volta trattato. (Una versione in codice assemblatore del calcolo della somma degli elementi dispari è presentata nell'illustrazione in alto a pagina 45.)

Nel pianificare la soluzione di un problema è improbabile che si pensi in termini di registri e di indirizzi di memoria; è il problema stesso a suggerire la notazione appropriata. Se il problema riguarda la fisica, il progetto di un programma può prendere le mosse da un'equazione come *F = ma;* in un contesto commerciale, la formula prescelta potrebbe essere *profitti = ricavi − spese*. Le operazioni specificate dalla formula poi sono tradotte in istruzioni esplicite alla macchina. I primi programmatori si resero conto che anche questa forma di traduzione poteva essere meccanizzata e in questa idea sta il germe concettuale di linguaggi come il FORTRAN, il BASIC e il Pascal. Per anni i linguaggi di questo tipo sono stati chiamati linguaggi di alto livello, ma oggi sembrerebbe più corretto chiamarli linguaggi di programmazione, perché i codici macchina e assemblatore non si possono classificare veramente come linguaggi.

A partire dal 1960 la maggior parte del software è stata scritta con l'ausilio dei linguaggi di programmazione, perché presentano molti vantaggi rispetto alle rappresentazioni di livello inferiore. Un'istruzione può corrispondere a molte istruzioni di macchina, e quindi i programmi tendono a essere più corti, con una corrispondente riduzione della fatica necessaria per scriverli e un parallelo miglioramento sul piano della chiarezza. Lavorare con concetti pertinenti al problema anziché con quelli definiti dalla macchina riduce inoltre le possibilità di errore; infine si apre la possibilità di una «indipendenza dalla macchina», cioè la possibilità di scrivere un unico programma che possa essere eseguito su molti calcolatori.

È importante distinguere fra un linguaggio di programmazione e una implementazione del linguaggio. Il linguaggio in sé è la notazione, l'insieme delle regole che definiscono la sintassi di un programma valido. L'implementazione di un linguaggio è un programma che trasforma la notazione di alto livello in successioni di istruzioni di macchina.

Vi sono due tipi principali di implementazione di un linguaggio: i compilatori e gli interpreti. Un compilatore traduce tutto il testo di un programma di alto livello in un processo unico, creando un programma completo in codice macchina che poi può essere eseguito indipendentemente dal compilatore. Lavorare con un linguaggio compilato in genere comporta tre stadi: la creazione del testo con un programma di redazione oppure con un programma di elaborazione testi, poi la compilazione del programma e infine l'esecuzione del programma compilato. Il termine compilatore è stato coniato nel 1951 da Grace Murray Hopper, allora alla Remington-Rand Univac, per descrivere il suo primo programma traduttore. All'interno del processo di traduzione, il programma estraeva successioni standard di istruzioni di macchina da tabelle memorizzate su nastro magnetico e le compilava in un programma completo.

Un interprete esegue un programma una istruzione alla volta, trasformando ogni costrutto di alto livello nelle corrispondenti istruzioni di macchina, sul momento. La differenza tra compilatore e interprete è analoga alla differenza fra

```
: SOMMADISPARI
  0 SWAP 0
  DO
     SWAP DUP 2 MOD
       IF +
       ELSE DROP
       THEN
  LOOP
;
```

23 34 7 9 4 SOMMADISPARI.

PAROLA	CATASTA							COMMENTO
23	23							
34	23	34						
7	23	34	7					
9	23	34	7	9				
4	23	34	7	9	4			
SOMMADISPARI	23	34	7	9	4			Chiama SOMMADISPARI.
0	23	34	7	9	4	0		
SWAP	23	34	7	9	0	4		
0	23	34	7	9	0	4	0	
┌DO	23	34	7	9	0			Toglie i valori del controllo di ciclo.
SWAP	23	34	7	0	9			
DUP	23	34	7	0	9	9		
2	23	34	7	0	9	9	2	
MOD	23	34	7	0	9	1		
IF	23	34	7	0	9			TOS = 1; esegue da IF a ELSE.
+	23	24	7	9				
ELSE	23	34	7	9				Salta dopo THEN.
DROP	23	34	7	9				Saltato.
THEN	23	34	7	9				Saltato.
└LOOP	23	34	7	9				Torna a DO.
┌DO	23	34	7	9				
SWAP	23	34	9	7				
DUP	23	34	9	7	7			
2	23	34	9	7	7	2		
MOD	23	34	9	7	1			
IF	23	34	9	7				TOS = 1; esegue da IF a ELSE.
+	23	34	16					
ELSE	23	34	16					Salta oltre THEN.
DROP	23	34	16					Saltato.
THEN	23	34	16					Saltato.
└LOOP	23	34	16					Torna a DO.
┌DO	23	34	16					
SWAP	23	16	34					
DUP	23	16	34	34				
2	23	16	34	34	2			
MOD	23	16	34	0				
IF	23	16	34					TOS = 0; esegue da ELSE a THEN.
+	23	16	34					Saltato.
ELSE	23	16	34					
DROP	23	16						
THEN	23	16						
└LOOP	23	16						Torna a DO.
┌DO	23	16						
SWAP	16	23						
DUP	16	23	23					
2	16	23	23	2				
MOD	16	23	1					
IF	16	23						TOS = 1; esegue da IF a ELSE.
+	39							
ELSE	39							Salta oltre THEN.
DROP	39							Saltato.
THEN	39							Saltato.
└LOOP	39							Fine delle iterazioni.
;	39							Ritorno da SOMMADISPARI.
.	< catasta vuota>							Stampa il risultato.

Il programma in Forth per la somma dei numeri dispari in un vettore non dichiara variabili né altre strutture di dati, ma lavora solo con valori in una «catasta di tipo *pushdown*». Quando viene chiamata *SOMMADISPARI*, gli elementi del vettore debbono essere nella catasta, con il numero degli elementi posto in cima. La riga sotto la definizione della procedura verrebbe battuta per eseguire il programma con un vettore di quattro elementi. È data poi una traccia completa dell'esecuzione del programma, che mostra il contenuto della catasta dopo l'esecuzione di ogni parola. Una «parola» numerica come 0 o 2 mette il numero sulla catasta; *SWAP* scambia i due elementi in cima; *DUP* mette sulla catasta una copia dell'elemento in cima (lo duplica); *DROP* elimina l'elemento in cima. Operatori come «+» e *MOD* sostituiscono i due elementi in cima alla catasta (che chiameremo TOS, dall'inglese Top Of the Stack) con il risultato. Il ciclo instaurato da *DO* toglie due elementi (*i* e *j*, per esempio) dalla catasta ed esegue le parole fino a *LOOP* per un totale di *i − j* volte. Il condizionale *IF* esegue le parole comprese fra *IF* ed *ELSE* quando la cima della catasta è diversa da zero; altrimenti esegue le parole fra *ELSE* e *THEN*.

traduttore di un'opera letteraria e interprete simultaneo. Il traduttore prende un manoscritto completo e fornisce un nuovo testo in un'altra lingua; l'interprete simultaneo traduce ogni espressione o frase nel momento in cui la si pronuncia. In effetti la maggior parte degli interpreti effettua qualche elaborazione iniziale del testo prima che cominci l'esecuzione: le parole chiave sono trasformate in simboli più brevi e i nomi delle variabili sono sostituiti da indirizzi. Tuttavia i due tipi di implementazione restano distinti: perché un programma interpretato possa essere eseguito, l'interprete deve essere presente nella memoria principale, mentre una volta che un programma è stato compilato la presenza del compilatore non è più necessaria.

In linea di principio qualunque linguaggio di programmazione potrebbe essere o interpretato o compilato, ma nella maggior parte dei casi la pratica ha fatto sì che una delle due implementazioni sia più comune dell'altra, per i singoli linguaggi. Il FORTRAN, il COBOL e il Pascal in genere sono compilati; il Logo, il Forth e l'APL sono quasi sempre interpretati. Il BASIC e il Lisp sono ampiamente disponibili in ambedue le forme. Il vantaggio principale della compilazione è la velocità; un interprete deve determinare una successione adatta di istruzioni ogni volta che viene eseguita un'istruzione, e pertanto un programma scritto in un linguaggio interpretato è quasi inevitabilmente più lento. D'altra parte un linguaggio interpretato è spesso più comodo per il programmatore: si attaglia bene a uno stile interattivo di sviluppo dei programmi. Parti di programma possono essere scritte, verificate e mandate in esecuzione senza mai lasciare l'interprete e, quando si identifica un errore, lo si può correggere immediatamente, senza dover tornare a un programma di redazione o elaborazione di testi ed effettuare poi una nuova compilazione del programma.

```
    ∇ SOMMA ← SOMMADISPARI TERMINI
[1] ∇ SOMMA ← +/(2 | TERMINI)/TERMINI

SOMMADISPARI 23 34 7 9

                TERMINI ← 23  34    7   9   Assegnamento iniziale valori.
        (2 | TERMINI) ←  1   0    1   1   Matrice di resti.
   (2 | TERMINI) /TERMINI ← 23        7   9   Compressione delle due matrici.
  +/(2 | TERMINI) /TERMINI ← 23   +   7 + 9   Riduzione mediante somma.
                  SOMMA ← 39                 Assegnamento del risultato.
```

Il programma in APL calcola la somma degli elementi dispari in un vettore con una funzione specificata in una riga. La funzione ha un parametro, *TERMINI*, un vettore che «sa» quanti elementi possiede, cosicché non occorre far comparire *N* nel programma. Un'istruzione di APL viene eseguita da destra verso sinistra, fuorché dove le parentesi modificano l'ordine di valutazione. In questo esempio si valuta prima l'espressione (2|*TERMINI*); essa calcola il resto dopo la divisione di ciascun elemento di *TERMINI* per 2 e crea una matrice della stessa dimensione di *TERMINI* per inserirvi i resti. Il simbolo «/» può indicare due operazioni distinte, ambedue presenti nell'esempio. Nell'espressione (2|*TERMINI*)/*TERMINI*, «/» è un operatore di «compressione» che crea un nuovo vettore in cui ogni elemento di *TERMINI* compare solo se il corrispondente elemento di (2|*TERMINI*) è diverso da zero. Nel simbolo «+/», «/» è un operatore di «riduzione» che riduce il vettore a un solo numero inserendo un «+» fra ogni coppia di elementi.

```
(DEFUN SOMMADISPARI
  (LAMBDA (TERMINI)
    (COND
      ((NULL TERMINI) 0)
      ((ODD(CAR TERMINI)) (PLUS (CAR TERMINI) (SOMMADISPARI(CDR TERMINI))))
      (T(SOMMADISPARI(CDR TERMINI))))))

(SOMMADISPARI '(23 34 7 9))

(SOMMADISPARI '(23 34 7 9))
  = (PLUS 23 (SOMMADISPARI '(34 7 9)))
    = (PLUS 23 (SOMMADISPARI '(7 9)))
      = (PLUS 23 (PLUS 7 (SOMMADISPARI '(9))))
        = (PLUS 23 (PLUS 7 (PLUS 9 (SOMMADISPARI '( )))))
      = (PLUS 23 (PLUS 7 (PLUS 9 0)))
    = (PLUS 23 (PLUS 7 9))
  = (PLUS 23 16)
= 39
```

Il programma in Lisp calcola la somma degli elementi dispari per mezzo di una funzione che richiama se stessa ricorsivamente. Una funzione in Lisp è una lista, in cui il primo elemento *(CAR)* è il nome della funzione e il resto della lista *(CDR)* è costituito dai parametri. *DEFUN* è una funzione di definizione di funzioni e *LAMBDA* precede i nomi dei parametri; qui l'unico parametro è la lista di numeri *TERMINI*. *COND* è una funzione condizionale che valuta il *CAR* delle liste che costituiscono i suoi parametri. Se il risultato è *T* (*true*, vero), si valuta il *CDR* della lista; altrimenti *COND* procede con l'esame della lista successiva. Qui vi sono tre possibilità. Se *TERMINI* è una lista vuota, *NULL* è vero e *SOMMADISPARI* dà valore zero. Se il *CAR* di *TERMINI* è dispari, si somma al totale corrente e viene chiamata *SOMMADISPARI* per valutare il *CDR* di *TERMINI*. Se nessuna di queste condizioni è vera, si raggiunge la clausola *T* (che deve essere vera); essa provoca semplicemente la chiamata di *SOMMADISPARI* con *(CDR(TERMINI))* come parametro. Durante ogni chiamata i calcoli restano sospesi.

Il funzionamento interno di un compilatore o di un interprete è un argomento troppo vasto per poterlo trattare qui, ma possiamo descrivere, almeno in generale, la struttura di un compilatore tipico. Nel processo di compilazione vi sono almeno tre fasi. La prima è l'analisi lessicale, in cui il compilatore identifica i vari simboli nel testo del programma e li classifica come parole chiave, valori numerici, nomi di variabili e così via. La fase successiva è quella dell'analisi sintattica *(parsing)*, in cui il compilatore determina le relazioni sintattiche fra le parole chiave e costruisce una rappresentazione della struttura del programma. Ogni *if*, per esempio, è associato con un *then* successivo. Nella terza fase si genera il codice macchina corrispondente alla struttura analizzata sintatticamente. Alcuni compilatori hanno anche una quarta fase, di ottimizzazione, in cui il codice viene riveduto per migliorarne l'efficienza.

Negli ultimi trent'anni la realizzazione dei compilatori è stata fatta oggetto di ricerche attente, e oggi esiste una metodologia ben organizzata per la loro costruzione. Il primo passo è quello di definire il linguaggio stesso in una forma completamente esplicita. Oggi è pratica comune specificare la grammatica in termini di «regole di produzione» che possono essere applicate ricorsivamente per generare tutte le possibili istruzioni del linguaggio. La creazione del linguaggio compilatore è poi un compito di programmazione relativamente immediato: esistono addirittura compilatori di compilatori che possono automatizzarlo almeno in parte.

L'idea di un linguaggio di programmazione è nata quasi contemporaneamente alla comparsa dei primi grandi calcolatori digitali. Nel 1945 il matematico tedesco Konrad Zuse inventò una notazione a cui diede il nome di Plankalkül. Le istruzioni del linguaggio avevano un formato bidimensionale: le variabili e i loro indici erano allineati verticalmente e le operazioni su di essi erano disposte lungo l'asse orizzontale. Zuse scrisse sulla carta programmi in Plankalkül (anche uno che simulava mosse del gioco degli scacchi) ma non implementò il linguaggio. Molte sue idee, comunque, sono state introdotte nei linguaggi moderni.

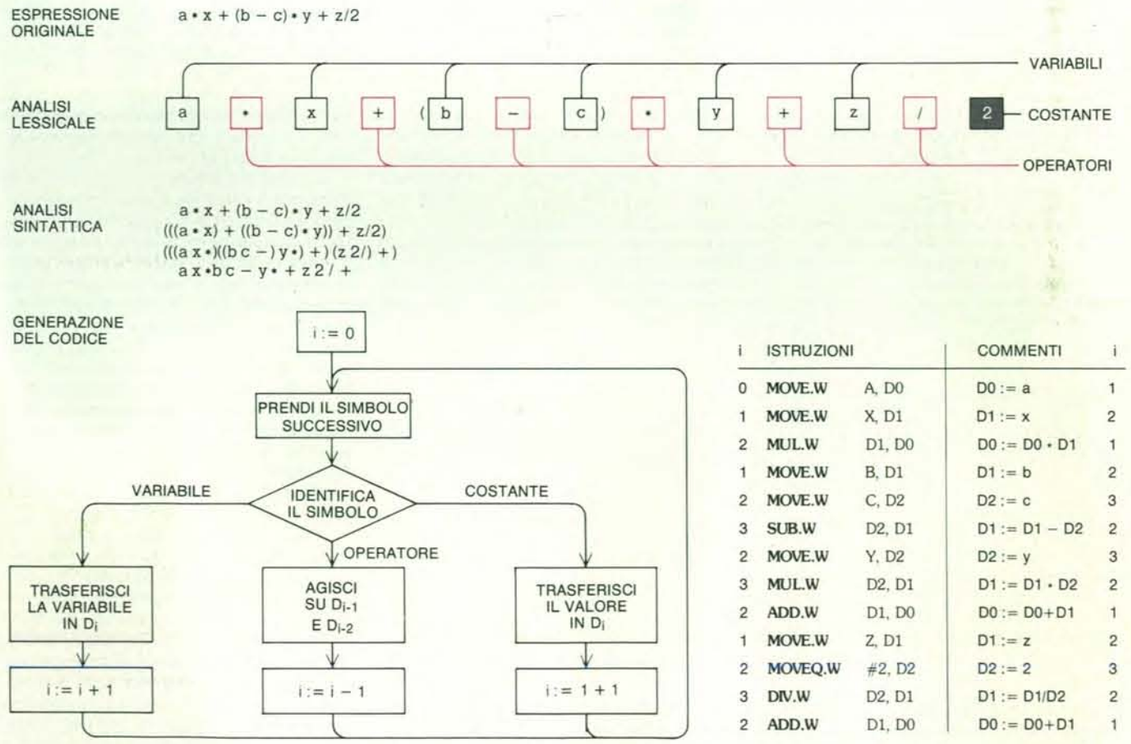
Sicuramente fra tutti i linguaggi di programmazione quello che ha esercitato l'influenza maggiore è il FORTRAN, sviluppato fra il 1954 e il 1957 da John Backus e collaboratori alla IBM. Il nome sta per *FORmula TRANslation* (traduzione di formule) e il linguaggio era studiato per i calcoli scientifici e numerici, attività per la quale è ancora largamente usato. All'epoca, il progetto fu guardato con notevole scetticismo. Allora i calcolatori erano una risorsa scarsa e di grande valore e, di conseguenza, si poneva fortemente l'accento sull'efficienza dei programmi. Si riteneva che un linguaggio di alto livello avrebbe inevitabilmente compromesso l'efficienza, ma Backus e il suo gruppo riuscirono a ottenere un risultato straordinario: crearono infatti un compilatore che produce-

CODICE MACCHINA	CODICE ASSEMBLATORE: ETICHETTE	ISTRUZIONI		COMMENTI
00100100 01011111	SOMMADISPARI	MOVE.L	(A7)+,A2	Porta l'indirizzo di ritorno da catasta a A2.
00100010 01011111		MOVE.L	(A7)+,A1	Porta l'indirizzo del primo elemento in A1.
00110010 00011111		MOVE.W	(A7)+,D1	Estrae n e lo mette in D1.
01000010 01000010		CLR.W	D2	Assegna valore 0 alla somma in D2.
01001110 11111010 00000000 00001110		JMP	COUNT	Salta a fine ciclo e controlla se n = 0.
00001000 00101001 00000000 00000000 00000000 00000001	LOOP	BTST	0,1(A1)	Se il termine indirizzato tramite A1 è pari...
01100111 00000010		BEQ.S	NEXT	...va a NEXT
11010100 01010001		ADD.W	(A1),D2	...altrimenti lo aggiunge alla somma in D2.
01010100 01001001	NEXT	ADDQ.W	#2,A1	Mette in A1 l'indirizzo del termine seguente.
01010001 11001001 11111111 11110010	COUNT	DBF	D1,LOOP	Decrementa D1; se non è −1, va a LOOP.
00111110 10000010		MOVE.W	D2,−(A7)	Porta la somma da D2 sulla catasta.
01001110 11010010		JMP	(A2)	Va all'indirizzo di ritorno.

Il codice macchina e il codice assemblatore specificano i passi del calcolo degli elementi dispari in termini delle risorse fisiche del calcolatore. Il codice è necessariamente specifico per una particolare macchina, in questo caso il microelaboratore Motorola 68000. L'algoritmo impiegato è molto simile alla procedura *SommaDispari* del Pascal, benché sia più compatto del codice che sarebbe generato da un compilatore Pascal. I parametri sono passati alla procedura su una catasta e il risultato è riportato sulla catasta; sulla catasta si trova anche l'indirizzo a cui deve ritornare l'esecuzione a fine procedura. La versione del programma in codice assemblatore, in cui le istruzioni prendono la forma di abbreviazioni «mnemoniche», può essere tradotta direttamente nel codice binario di macchina eseguito dal microelaboratore.

va un codice qualitativamente pari a quello dei programmi codificati a mano.

Più o meno nello stesso periodo, Grace Hopper e collaboratori alla Remington-Rand Univac svilupparono un linguaggio di programmazione, chiamato Flow-Matic, per l'elaborazione di dati commerciali. Era meno raffinato del FORTRAN, ma l'esperienza che consentì di accumulare nel giro di vari anni costituì la fonte principale di ispirazione per il COBOL. Un altro linguaggio fondamentale introdotto verso la fine degli anni cinquanta fu l'Algol (ovvero *Algorithmic language*, linguaggio algoritmico). L'Algol-58, la prima versione, fu progettato da un comitato internazionale che si basò tanto sulla sintassi pragmatica del FORTRAN quanto sul-



i	ISTRUZIONI		COMMENTI	i
0	MOVE.W	A, D0	D0 := a	1
1	MOVE.W	X, D1	D1 := x	2
2	MUL.W	D1, D0	D0 := D0 • D1	1
1	MOVE.W	B, D1	D1 := b	2
2	MOVE.W	C, D2	D2 := c	3
3	SUB.W	D2, D1	D1 := D1 − D2	2
2	MOVE.W	Y, D2	D2 := y	3
3	MUL.W	D2, D1	D1 := D1 • D2	2
2	ADD.W	D1, D0	D0 := D0+D1	1
1	MOVE.W	Z, D1	D1 := z	2
2	MOVEQ.W	#2, D2	D2 := 2	3
3	DIV.W	D2, D1	D1 := D1/D2	2
2	ADD.W	D1, D0	D0 := D0+D1	1

Un compilatore (o programma traduttore) per un linguaggio di programmazione è articolato in almeno tre stadi, i quali sono qui illustrati per il caso, particolarmente semplice, di una espressione aritmetica in Pascal. Nell'analisi lessicale i simboli che costituiscono il programma sono identificati e classificati. L'analisi sintattica ricostruisce le relazioni sintattiche fra i simboli. In una espressione aritmetica il compito principale dell'analizzatore sintattico consiste nel determinare quali operandi siano associati con ogni operatore: qui lo si effettua confrontando la precedenza degli operatori adiacenti (supponendo che la moltiplicazione preceda la divisione, che questa preceda l'addizione, e così via); vengono aggiunte parentesi a racchiudere le operazioni di precedenza maggiore. L'espressione è trasformata in notazione «postfissa» mediante lo scambio dell'operatore e del secondo operando in ogni sottoespressione. A questo punto le parentesi vengono eliminate, il che porta a un'espressione che può essere valutata da sinistra a destra. La generazione del codice trasforma l'espressione analizzata sintatticamente in istruzioni di macchina, mediante un semplice algoritmo per l'assegnazione di ogni variabile a un registro fisico.

la notazione, più elegante, del Plankalkül. Il risultato fu un linguaggio al contempo leggibile e pratico, che ha avuto un posto importante nella genealogia di linguaggi successivi, fra i quali il Pascal.

Numerosi altri linguaggi affondano le loro radici nello stesso periodo. Il COMIT fu creato per l'analisi di testi, e l'APT per il controllo di macchine utensili. Il JOVIAL, derivato dall'Algol, fu il primo linguaggio multiuso di ampia diffusione, adatto per applicazioni in campo scientifico e commerciale. Agli inizi degli anni sessanta comparvero il Lisp e la notazione (ma non ancora l'implementazione) dell'APL.

Il rapido proliferare dei linguaggi ha messo in difficoltà molti. In fin dei conti, la maggior parte della matematica si fa con un'unica notazione, universalmente accettata. Implementare un nuovo linguaggio è un'impresa di tutto rispetto, e per un programmatore acquisire familiarità con un nuovo linguaggio richiede tempo. Presto vennero inaugurati vari progetti di ricerca per la costruzione di un nuovo linguaggio tanto completo e versatile da poter fungere da linguaggio universale della programmazione. Tutti i tentativi sono falliti. Il successo parziale del PL/I, sviluppato con il finanziamento della IBM nel 1965, ha messo bene in chiaro come un linguaggio per tutti gli scopi sia sempre, con tutta probabilità, difficile da imparare non meno che da implementare. Inoltre, con il diversificarsi delle tecniche della computazione, ci si è resi conto che continueranno a rendersi necessari nuovi linguaggi per soddisfare aree applicative particolari.

In un certo senso, si può dire che tutta la ricerca sui linguaggi di programmazione a partire dal 1957 sia stata motivata dal tentativo di correggere i punti deboli del FORTRAN. In effetti, lo stesso FORTRAN è stato sottoposto più volte a revisione. La versione originale poneva al programmatore vincoli arbitrari (per esempio, il nome di una variabile non poteva essere lungo più di sei caratteri) e offriva strumenti limitati per la definizione di strutture di dati. Le lacune più gravi, però, stavano forse negli strumenti per il controllo del flusso del programma. Tutti i punti di diramazione dovevano essere definiti da numeri di riga e se non si prestava grande attenzione il funzionamento di un programma poteva essere oscurato da un groviglio di istruzioni *goto*. Versioni successive hanno introdotto strutture di controllo che incoraggiano uno stile di programmazione più leggibile.

Tutti i linguaggi di cui ho parlato fin qui possono essere definiti procedurali, o imperativi. Un programma scritto in un linguaggio del genere dice come si ottiene un risultato: prima fai questo, poi fai questo, poi quest'altro. Esistono però anche linguaggi non procedurali, o descrittivi, la cui importanza va gradualmente crescendo. Un programma descrittivo dice quale sia il risultato desiderato, senza specificare come raggiungerlo. Il programma definisce relazioni, anziché il flusso di controllo, e così il programmatore è liberato dalla responsabilità di elaborare i passi di un algoritmo e di specificarne l'ordine.

I linguaggi non procedurali più appariscenti sono i programmi di tabellone elettronico come il VisiCalc e il MultiPlan, diventati popolari con la diffusione dei calcolatori personali. In MultiPlan un calcolo è specificato scrivendo formule, più o meno come in BASIC o in FORTRAN, ma l'ordine in cui le formule sono valutate è determinato dalla implementazione del linguaggio anziché dal programmatore. In una certa misura le relazioni temporali sono sostituite da relazioni spaziali. In un linguaggio convenzionale il valore prodotto in uscita da una procedura può fungere da ingresso per la procedura successiva; il concetto analogo in un tabellone elettronico fa dipendere il valore di una cella dal valore di un'altra.

Il concetto di procedura ha ancora meno senso nel linguaggio Prolog, derivato dal Lisp e che, negli ultimi anni, ha attirato l'attenzione di molti ricercatori nel campo dell'intelligenza artificiale. Il linguaggio è costituito esclusivamente da dichiarazioni, senza istruzioni. Così la relazione *(prodotto altezza larghezza area)* descrive l'uguaglianza *area = altezza × larghezza,* ma non specifica che l'altezza e la larghezza sono le grandezze date o che debba essere calcolata l'area. La stessa relazione serve per trovare l'altezza quando sono note area e misura della base.

Un'altra tendenza nell'evoluzione dei linguaggi di programmazione è la crescita di interesse per i sistemi notazionali chiamati linguaggi orientati all'oggetto. Come abbiamo detto, le istruzioni della maggior parte dei linguaggi di programmazione sono imperative: l'entità a cui ci si rivolge non è nominata semplicemente perché c'è un'unica possibilità, la figura astratta del calcolatore come un tutto. In un linguaggio orientato all'oggetto il calcolatore è diviso concettualmente in oggetti a cui ci si può indirizzare individualmente. Inoltre, gli oggetti possono comunicare l'uno con l'altro inviandosi messaggi.

L'idea di oggetti di software è stata introdotta da Ole-Johan Dahl e Kristen Nygaard del Centro di calcolo norvegese di Oslo nel Simula 67, un linguaggio derivato dall'Algol 60. La loro idea però non ha guadagnato molta attenzione fino a che, negli anni settanta, Alan Kay e un gruppo di colleghi (tra i quali anch'io, a quell'epoca) al Palo Alto Research Center (PARC) della Xerox non svilupparono il linguaggio Smalltalk. Lo Smalltalk è costituito esclusivamente da costruzioni orientate a oggetti, il che rende la specificazione del linguaggio molto concisa e molto generale; d'altra parte, poiché nel linguaggio qualunque cosa è un oggetto, alcuni importanti meccanismi di strutturazione dei dati non possono essere realizzati in modo efficiente.

Un oggetto di software è costituito da strutture di dati e algoritmi. Ogni oggetto «sa» come eseguire operazioni sui suoi stessi dati, ma per il resto del programma quell'oggetto può essere trattato come una scatola nera il cui funzionamento interno è irrilevante. In effetti, oggetti diversi possono usare algoritmi molto diversi per eseguire compiti che il programmatore identifica con la medesima parola chiave. Proprio come i pinguini, i cavalli e i millepiedi usano metodi palesemente diversi per realizzare quell'attività che identifichiamo in generale come locomozione, gli oggetti i cui dati sono costituiti da interi, matrici e numeri complessi possono usare metodi diversi per eseguire l'operazione di addizione.

I miei colleghi e io alla Apple Computer, Inc., abbiamo sviluppato un linguaggio, chiamato Clascal, che aggiunge il concetto di classi di oggetti alla struttura di fondo del Pascal. Il Clascal, lo Smalltalk, il Simula e alcuni altri linguaggi orientati all'oggetto consentono agli oggetti di una classe di ereditare proprietà da una sopraclasse a cui appartengono, cosicché le singole classi non debbono necessariamente essere costruite da zero; debbono essere specificati solo quei tratti che distinguono le singole classi. I pinguini, i cavalli e i millepiedi hanno in comune il concetto di arto; si differenziano per il numero degli arti e i particolari del metodo di locomozione. L'eredità è un altro meccanismo di astrazione, che consente l'utilizzazione delle proprietà di una classe da parte di molte sottoclassi.

L'eredità si rivela particolarmente utile nella creazione di software grafico interattivo, un altro campo in cui oggi fervono molte iniziative. Con immagini grafiche si possono costruire interi linguaggi di programmazione. In effetti, anche certi giochi per calcolatore che si basano fortemente sulla grafica presentano alcune caratteristiche di un linguaggio di programmazione. Un esempio degno di nota è un gioco che viene chiamato Robot Odissey I, introdotto recentemente dalla Learning Company: «robot» programmati collegando porte logiche elettroniche e altri componenti su uno schermo video possono incorporare i concetti di esecuzione condizionale e di definizione di procedura. Un vero e proprio linguaggio di programmazione visivo, chiamato provvisoriamente Mandala, è in corso di sviluppo alla VPL Research di Palo Alto, a opera di Jaron Z. Lanier e collaboratori. Un esempio di come possa apparire un programma in Mandala si può vedere sulla copertina di questo fascicolo.

Un'altra direzione di sviluppo dei linguaggi di programmazione è lo sfruttamento dell'elaborazione in parallelo in sistemi costituiti da più unità di elaborazione. Sembrerebbe che 100 unità di elaborazione debbano poter risolvere un problema a una velocità 100 volte superiore rispetto a un elaboratore con la stessa velocità intrinseca, ma il guadagno di velocità può essere ottenuto solo se il software è in grado di suddividere il problema in molti segmenti che possano essere elaborati simultaneamente.

Alcuni linguaggi forniscono un meccanismo esplicito per indicare compiti che possono essere effettuati in parallelo: un esempio è costituito dal linguaggio Occam, sviluppato dalla Inmos, una società inglese che produce semiconduttori. Altri linguaggi lasciano al compilatore l'analisi del programma e l'identificazione di possibilità di esecuzione in parallelo. Un linguaggio del genere è COMPEL *(«COMPute parallEL»),* alla cui realizzazione ho collaborato nel 1969 con Horace J. Enea. Un programma in COMPEL è costituito esclusivamente da istruzioni di assegnamento, che non sono eseguite necessariamente nella successione in cui sono scritte: starebbe al compilatore dedurre quali calcoli debbano essere portati a termine per primi. Non è mai stato scritto un compilatore per COMPEL, ma da allora sono stati effettivamente implementati linguaggi con meccanismo analogo (sono i cosiddetti linguaggi a flusso di dati).

I linguaggi di programmazione sono molto diversi fra loro, ed è impossibile stilarne una sorta di graduatoria su un'unica scala. Non esiste un linguaggio di programmazione migliore di tutti gli altri, esattamente come non esiste una lingua naturale migliore di tutte le altre. «Parlo spagnolo a Dio, italiano alle donne, francese agli uomini e tedesco al mio cavallo», diceva Carlo V (presumibilmente in francese). Anche un linguaggio di programmazione va scelto in funzione dell'obiettivo che ci si prefigge.

add(Adamo genitore-di Caino)
add(Adamo genitore-di Abele)
add(Eva genitore-di Caino)
add(Eva genitore-di Abele)
add(Caino genitore-di Enoch)

which(x:x genitore-di Abele)
Adamo
Eva
No (more) answers

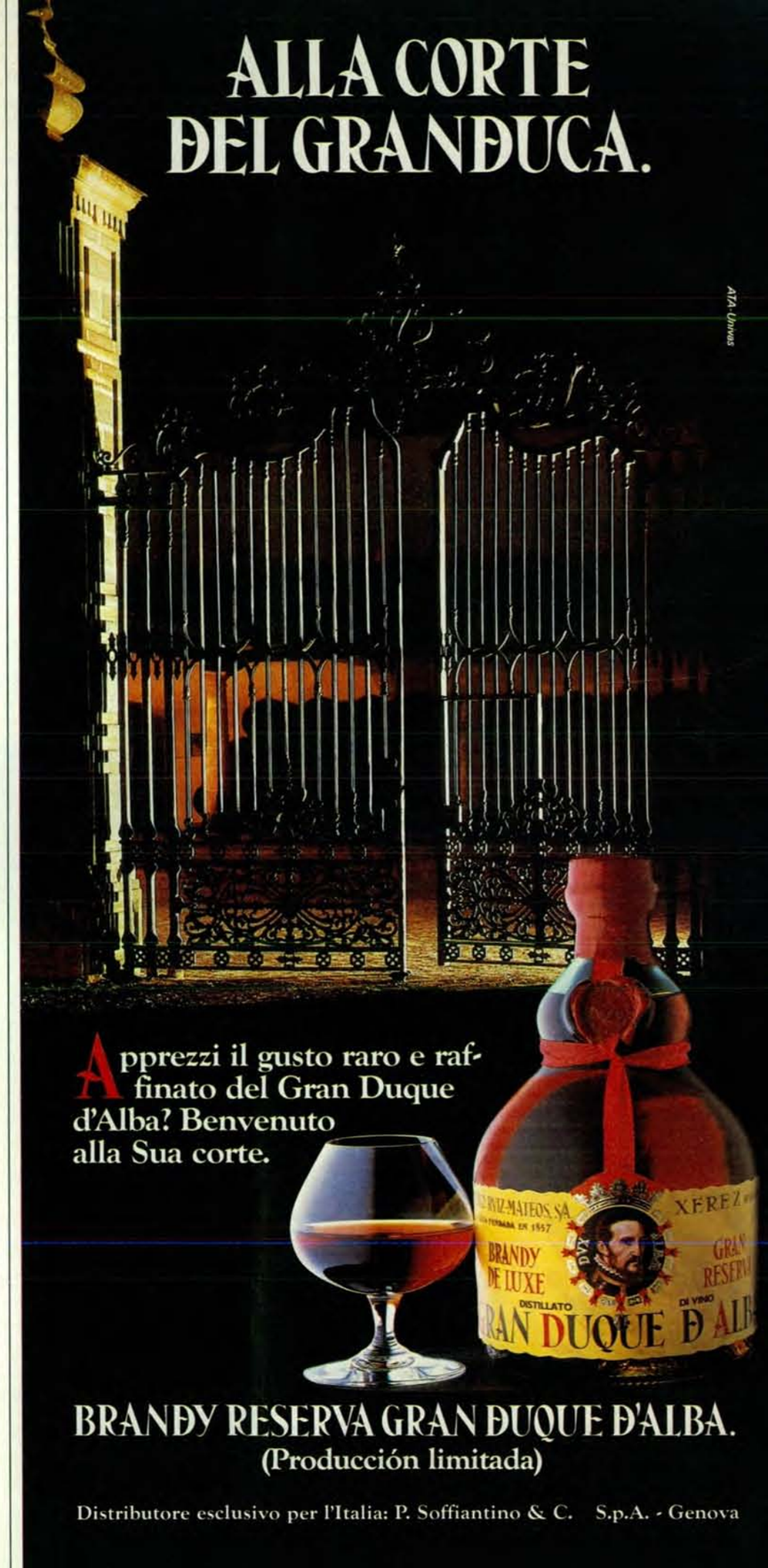
which(x : Eva genitore-di x)
Caino
Abele
No (more) answers

add(x progenitore-di y **if** x genitore-di y)
add(x progenitore-di y **if** z genitore-di y e x genitore-di z)

which(x:x progenitore-di Enoch)
Caino
Adamo
Eva
No (more) answers

which(x: Adamo progenitore-di x)
Caino
Abele
Enoch
No (more) answers

Il Prolog, un linguaggio non procedurale, non contiene istruzioni ma è formato solo da dichiarazioni. In altre parole, un programma in Prolog non dà istruzioni esplicite per eseguire un'operazione, ma si limita a indicare relazioni e a trarre deduzioni da esse. L'illustrazione mostra un programma in un dialetto del Prolog, il Micro-Prolog. Le prime cinque dichiarazioni presentano relazioni genitore-figlio. Il sistema può poi rispondere a domande relative ai fatti enunciati, per esempio identificando i genitori di Abele e i figli di Eva. Sono poi introdotte due regole di deduzione logica per definire la relazione «progenitore di» in termini della relazione «genitore di». Il sistema le può applicare per trovare tutti gli antenati o i discendenti di un individuo.



Sistemi operativi

Abbracciano molti livelli di complessità, dai comandi inseriti tramite tastiera ai particolari della commutazione elettronica, e sono organizzati secondo una gerarchia di astrazione crescente

di Peter J. Denning e Rober L. Brown

Se si batte il comando *data* su un terminale collegato a un sistema di elaborazione e poi si preme il tasto di *Ritorno,* quasi all'istante compare sullo schermo il messaggio *15 settembre 1984.* Chiedere che giorno è può sembrare una delle domande più semplici da porre a un calcolatore, eppure dà l'avvio a una complessa successione di eventi, che fanno intervenire molte fra le risorse hardware e software del sistema. La coordinazione degli eventi e la gestione delle risorse rientrano fra i compiti di cui è responsabile quell'insieme di programmi che viene chiamato «sistema operativo» del calcolatore. Il sistema operativo fornisce i mezzi e i servizi necessari a quasi tutti gli altri programmi.

Consideriamo gli eventi che debbono accadere affinché sia fornita risposta alla domanda sulla data. Nel momento in cui viene battuto un carattere del comando, la tastiera trasmette al calcolatore un codice, che viene ricevuto da una piastra circuitale che ha il compito di gestire le comunicazioni con il terminale. La piastra memorizza ciascun codice di carattere in una zona riservata della memoria, detta *buffer,* memoria di transito o memoria tampone, ed emette un segnale che «interrompe» l'unità centrale di elaborazione, attivando un programma chiamato pilota del terminale. Il pilota del terminale rinvia al terminale una copia del codice, per la visualizzazione sullo schermo.

Quando viene ricevuto il codice del tasto *Ritorno,* il che significa che la battitura del comando è compiuta, il pilota del terminale attiva un programma chiamato ascoltatore (poiché presta attenzione alle richieste degli utenti). L'ascoltatore legge i caratteri *d a t a* dalla memoria di transito della tastiera, esplora una memoria a disco magnetico cercandovi un programma chiamato *data,* carica questo programma nella memoria centrale e avvia la sua esecuzione. A sua volta il programma *data* consulta un orologio incorporato nei circuiti, il quale tiene conto dei millisecondi trascorsi da una data iniziale fissata. Sulla base di questo conteggio il programma calcola il giorno, il mese e l'anno e traduce queste informazioni nella successione di caratteri *15 settembre 1984.* Questa successione viene passata al programma pilota del terminale, il quale trasmette il codice binario di ciascun carattere al terminale, dove appare sullo schermo.

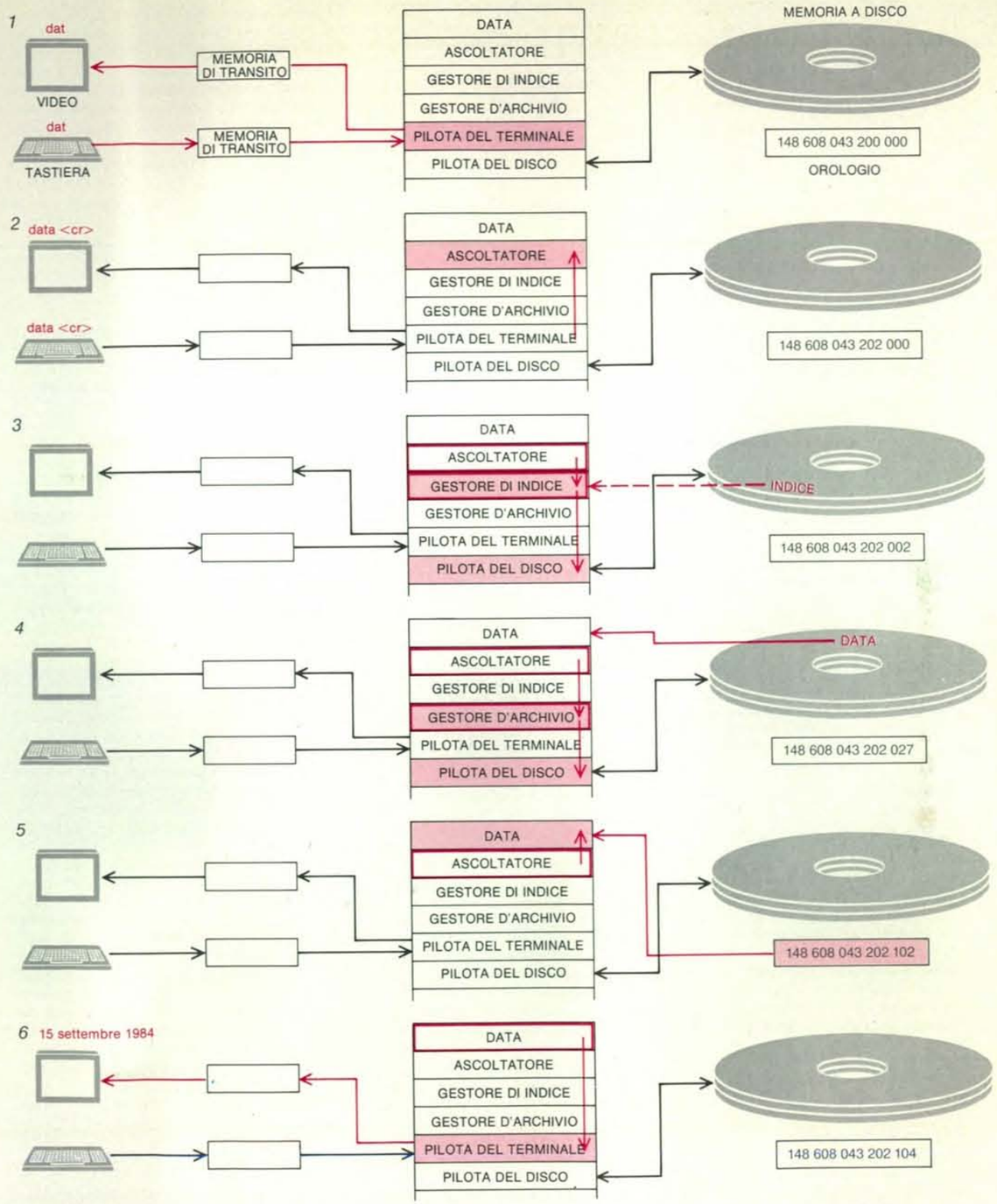
Sarebbe possibile descrivere ciascuno di questi eventi anche più minutamente. Per esempio, prima di poter caricare il programma *data,* l'ascoltatore deve esplorare un indice per trovare in quale zona del disco sia memorizzato il programma; in effetti, l'indice stesso dev'essere letto dal disco. Il disco è strutturato in piste concentriche, e ciascuna pista è suddivisa in settori; quindi debbono essere fornite istruzioni perché la testina di lettura si disponga sopra la pista giusta e i dati binari siano letti quando i settori scelti passano sotto la testina. La successione di bit che ne risulta viene immagazzinata provvisoriamente in una memoria di transito. Quando il programma viene caricato, è necessario assegnargli dello spazio nella memoria centrale; quando la sua esecuzione è terminata, quello spazio dev'essere recuperato. Questa successione di eventi è ancora più complicata se il calcolatore si occupa di parecchi programmi nello stesso tempo. In questo caso può accadere che uno dei programmi debba essere momentaneamente sospeso mentre l'unità centrale di elaborazione si occupa di un altro; dopo, il primo programma deve riprendere proprio come se non vi fosse mai stata alcuna interruzione.

Da questo esempio si può arguire che un sistema operativo abbraccia tutta la gamma di complessità che si trova in un sistema di elaborazione. Certe porzioni del sistema operativo interagiscono direttamente con i circuiti del calcolatore, nei quali certi eventi (per esempio la commutazione delle porte logiche elementari) si svolgono a una scala temporale piccolissima, di pochi miliardesimi di secondo. All'altro capo dello spettro, certe porzioni del sistema operativo comunicano con l'utente, il quale emette i suoi comandi a un ritmo molto più pacato, diciamo uno ogni qualche secondo. Quando si batte un tasto sul terminale, ciò può dar luogo a 10 chiamate per i programmi del sistema operativo, all'esecuzione di 1000 istruzioni macchina e a 10 milioni di cambiamenti di stato nelle porte logiche.

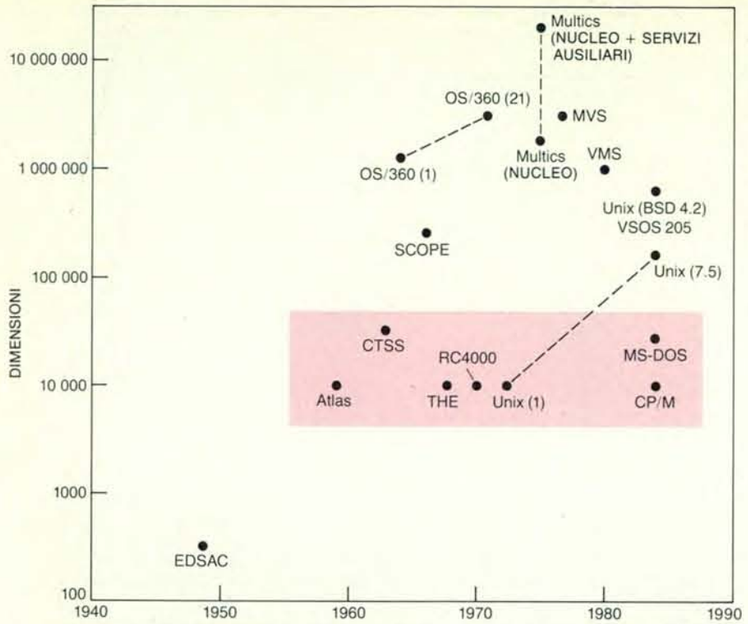
L'impostazione adottata per gestire tutta questa complessità si è dimostrata d'importanza cruciale virtualmente in tutte le aree della scienza del calcolatore. L'idea di fondo è quella di istituire una gerarchia di livelli di astrazione, tali che a ciascun livello sia consentito ignorare i particolari di ciò che accade a tutti i livelli inferiori. Così, quando carica un programma prelevandolo dalla memoria a disco, il programma ascoltatore non ha bisogno di specificare la posizione della testina di lettura: queste operazioni meccaniche sono compiute da un programma che sta a un livello gerarchico inferiore. Al livello più alto di tutti sta l'utente del sistema, il quale idealmente è isolato da tutto, tranne che dagli obiettivi che desidera conseguire.

I primi sistemi operativi furono allestiti per i primi calcolatori elettronici verso la fine degli anni quaranta. Si trattava di insiemi di semplici procedure d'ingresso e d'uscita, per esempio un programma per memorizzare in posizioni successive della memoria i codici binari letti da un nastro perforato. Il sistema operativo consisteva in tutto in poche centinaia di istruzioni macchina.

Verso la fine degli anni cinquanta la maggior parte dei calcolatori funzionava nel «modo a lotti» *(batch mode).* Un sistema operativo raccoglieva i programmi presentati da parecchi utenti e li eseguiva in rapida successione, eliminando così i ritardi provocati dal caricamento manuale di un programma alla volta. I sistemi operativi di questo tipo venivano chiamati supervisori o controllori. Oltre a svolgere la loro funzione principale di caricare i programmi, gestivano anche i dispositivi di memoria secondari (come i nastri, i tamburi e i dischi magnetici), assegnavano la memoria principale e si occupavano dell'ingresso e dell'uscita. Nella maggioranza dei casi comprendevano anche una «biblioteca» di procedure di uso frequen-



L'esecuzione di un comando dà l'avvio a eventi situati a parecchi livelli nella gerarchia dei programmi che costituiscono il sistema operativo. Questo comando è semplicemente una richiesta di data. Ogni carattere battuto sulla tastiera *(1)* è ricevuto da un programma pilota del terminale che lo rimanda allo schermo video. Quando viene inviato un segnale di ritorno carrello *(cr),* il pilota del terminale passa la successione di caratteri *d a t a* al programma ascoltatore *(2),* che l'interpreta come il nome di un comando. L'ascoltatore chiede al gestore dell'indice di cercare *data* nell'indice. Il gestore dell'indice chiede al pilota dell'unità a dischi di copiare l'indice in una memoria di transito o *buffer* entro lo spazio di memoria del gestore dell'indice *(3).* Quando il comando è stato trovato, l'ascoltatore fa sì che il gestore dell'archivio carichi nella memoria il codice binario del programma *data,* ricorrendo di nuovo al pilota dell'unità a dischi *(4).* L'ascoltatore attiva il programma *data* che legge un «orologio» *(5),* un dispositivo circuitale che tien conto dei milionesimi di secondo trascorsi da un istante iniziale fissato, qui la mezzanotte dell'1 gennaio 1980. A partire da questo numero, il programma calcola la data corrente e la mostra sul terminale video, mediante il pilota del terminale *(6).* Ascoltatore, piloti e gestori fanno parte del «nucleo» del sistema operativo; *data* è un programma ausiliario.



L'evoluzione dei sistemi operativi dà l'impressione che vi sia una tendenza verso la crescita esponenziale, ma di recente sono stati introdotti sistemi più piccoli per i microcalcolatori. La grandezza di un sistema operativo è misurata in unità che corrispondono alle «parole» della memoria di macchina o alle istruzioni del linguaggio di assemblatore. L'EDSAC è stato realizzato all'Università di Cambridge, l'Atlas all'Università di Manchester, il CTSS al Massachusetts Institute of Technology, il THE al Politecnico di Eindhoven e l'RC 4000 all'Università della Danimarca. Lo Scope è un prodotto della Control Data Corporation, come il VSOS 205; l'OS/360 e l'MVS sono prodotti della IBM, VMS è della Digital Equipment Corporation. Il Multics è stato realizzato in collaborazione dal MIT e dai Bell Laboratories, mentre l'Unix dai Bell Laboratories soltanto. Il CP/M e l'MS/DOS sono sistemi operativi per microcalcolatori, introdotti rispettivamente dalla Digital Research, Inc. e dalla Microsoft Corporation. I sistemi contenuti nel riquadro in colore sono nuclei per macchine singole, aventi probabilmente le minime dimensioni possibili.

te. Per esempio, in molte applicazioni del calcolatore è richiesto un ordinamento delle informazioni; se della biblioteca fa parte una procedura di ordinamento versatile, il sistema operativo può caricarla insieme con ogni programma che ne abbia necessità.

Verso il 1960 cominciarono a essere progettati i primi sistemi a divisione di tempo. Quando il calcolatore viene fatto funzionare in questo modo, l'attenzione dell'unità centrale di elaborazione viene spostata rapidamente fra parecchi programmi di utente, e ciò dà a tutti gli utenti l'illusione che i loro programmi vengano eseguiti simultaneamente. Nella costruzione di questi sistemi si dovettero affrontare i problemi connessi con la condivisione delle varie risorse: l'unità centrale, la memoria, i programmi. La soluzione di questi problemi favorì numerosi e importanti progressi concettuali, fra cui la sincronizzazione dell'elaborazione parallela, la memoria virtuale, lo svincolamento dell'ingresso e dell'uscita dalle apparecchiature particolari e i linguaggi di comando interattivi. Tutti questi punti saranno discussi nel seguito.

Via via che i sistemi operativi diventavano più elaborati, diventavano anche più grandi. Per esempio, il Compatible Time Sharing System (Sistema a divisione di tempo compatibile), messo a punto presso il Massachusetts Institute of Technology nel 1963, consisteva di circa 32 000 parole di memoria da 36 bit, mentre l'OS/360, introdotto un anno dopo dalla International Business Machines Corporation, possedeva più di un milione di istruzioni macchina. Il sistema Multics, approntato nel 1975 dal MIT e dai Bell Laboratories, aveva oltrepassato i 20 milioni di istruzioni.

A questo punto, tuttavia, cominciò a farsi sentire un effetto equilibratore: sul mercato erano comparsi i minicalcolatori e cominciavano a spuntare i microcalcolatori (compresi i calcolatori personali). Queste macchine erano più lente delle unità centrali di quel tempo e avevano una capacità di memoria minore, ma permettevano a una fascia molto più ampia di utenti potenziali di accostarsi all'elaborazione. Per costringere i sistemi operativi entro i limiti di capacità più ristretti dei mini o dei microcalcolatori, le funzioni del sistema furono suddivise. I servizi di cui quasi tutti i programmi hanno bisogno, per esempio le procedure d'ingresso e d'uscita, furono collocati in un «nucleo», che si trova nella memoria principale del calcolatore ogni volta che esso è in funzione. Altri programmi, chiamati servizi ausiliari, o utilità di sistema, sono memorizzati su disco e vengono letti e trasferiti nella memoria principale solo quando è necessario. A giudicare dai sistemi operativi introdotti negli ultimi anni, sembra che il nucleo minimo occorrente per gestire le risorse di un elaboratore singolo consista in poche decine di migliaia di istruzioni. I servizi ausiliari e le biblioteche di programmi, che crescono di continuo in modo quasi esponenziale, mettono a dura prova la capacità delle memorie secondarie.

L'evoluzione dei sistemi operativi non è certo terminata. Una nuova schiera di utenti, per molti dei quali l'informatica non è un'occupazione a tempo pieno, avanza nuove pretese nei confronti del *software*. Una risposta è stata la nascita di terminali video interattivi di tipo grafico. Ricorrendo a un terminale video di questo genere, si potrebbe cancellare un archivio non battendo il comando *cancella* sulla tastiera, bensì indicando il disegno di un cestino della carta straccia. Sono stati sviluppati anche nuovi assetti organizzativi per i sistemi di elaborazione: invece di avere un solo calcolatore di grandi dimensioni collegato con molti terminali, si può assegnare a ciascun utente una stazione di lavoro, che possiede un'unità di elaborazione propria e comunica con le altre stazioni di lavoro mediante una rete ad alta velocità. Spetta al software del sistema operativo coordinare le azioni dei vari calcolatori inseriti in una rete di elaborazione distribuita di tal fatta.

La struttura gerarchica di un sistema operativo moderno suddivide le sue funzioni a seconda della complessità, della scala temporale caratteristica e del livello di astrazione. L'illustrazione della pagina a fronte in basso mostra un'organizzazione distribuita su 13 livelli. Non si tratta dello schema di qualche sistema operativo particolare, bensì di un modello che accoglie idee proprie di parecchi sistemi; esso comprende strumenti per l'elaborazione distribuita. Ciascun livello gestisce un insieme di «oggetti», che possono avere carattere fisico oppure logico e la cui natura varia parecchio da un livello all'altro. Inoltre per ciascun livello sono definite le operazioni che possono essere eseguite su quegli oggetti.

I livelli più bassi comprendono l'hardware del sistema. Il primo livello è quello dei circuiti elettronici, e i suoi oggetti sono registri, celle di memoria, porte logiche e così via; le operazioni definite su questi oggetti sono azioni come l'azzeramento di un registro o la lettura di una locazione di memoria. Il secondo livello è quello dell'insieme delle istruzioni dell'unità di elaborazione; questo livello può trattare enti alquanto più astratti, per esempio una catasta (o pila) di valutazione (cioè una successione di registri e di celle di memoria dove certi valori numerici vengono custoditi mentre su di essi viene eseguita qualche operazione). Le operazioni di questo livello sono le istruzioni che può eseguire l'unità di elaborazione stessa, per esempio *somma, sottrai, carica* e *memorizza*.

Al terzo livello si aggiunge il concetto di procedura, o sottoprogramma, un frammento autosufficiente di programma che può essere chiamato durante l'esecuzione di un programma più ampio, al quale poi restituisce il controllo al punto in cui era stato chiamato. Il quarto livello introduce le interruzioni; un'interruzione fa sì che l'unità di elaborazione effettui una registrazione dello stato in cui si trova e poi si dedichi a un altro compito. Tra gli eventi che fanno scattare un'interruzione vi sono le situazioni di errore, per esempio lo straripamento *(overflow)* di un registro aritmetico, ma anche eventi più ordinari, come la ricezione di un codice di carattere da un terminale.

I primi quattro livelli corrispondono nel loro insieme all'ossatura fondamentale della macchina, così come la si riceve dal costruttore, per quanto vi siano strette interazioni con alcuni elementi del nucleo del sistema operativo. Per esempio le interruzioni sono generate da un componente circuitale, ma le procedure chiamate quando l'unità di elaborazione viene interrotta fanno parte del nucleo.

I concetti collegati esplicitamente con il coordinamento di più compiti compaiono per la prima volta al quinto livello, che viene chiamato il livello dei «processi primitivi», ovvero programmi singoli in corso di esecuzione. Poiché un processo primitivo può essere interrotto in ogni momento, occorre che vi sia un dispositivo che sospenda un processo e poi lo faccia riprendere. Questo dispositivo consiste nella «parola di stato», una struttura di dati che può accogliere il contenuto di tutti i registri dell'unità centrale di elaborazione, e in un'operazione di «commutazione del contesto». Quando un processo dev'essere sospeso, l'operazione di commutazione del contesto fa sì che i valori dei registri vengano copiati nella parola di stato corrispondente a quel processo; alla ripresa del processo essa ripristina nei registri i valori precedenti.

Se tutte le attività che si svolgono in un calcolatore fossero affatto indipendenti l'una dall'altra, occorrerebbe poco più del concetto di parola di stato per avere un sistema operativo per processi multipli. In realtà un processo dipende sovente dai risultati di un altro, sicché i programmi debbono essere sincronizzati. Un programma che abbia bisogno di dati contenuti in un archivio su disco, per esempio, non può procedere finché i dati non siano stati letti e trasferiti nella memoria principale. Ma il programmatore non può sapere in anticipo quanto durerà la lettura del disco, e quindi dev'esserci un modo per far sì che un programma aspetti finché un altro non segnala di essere pronto.

Il concetto che fornisce la chiave per la sincronizzazione è il semaforo, il quale occupa una posizione fondamentale nella teoria dei sistemi operativi. Nel caso più semplice, può essere d'aiuto considerare il semaforo come un dispositivo del tutto simile a un semaforo ferroviario, fornito di una luce verde e di una luce rossa che segnalano se un processo può continuare o meno in condizioni di «sicurezza». Nel punto in cui un processo dev'essere sincronizzato con qualche procedura esterna, il programmatore inserisce un'istruzione, per esempio *aspetta (semaforo A)*. Ogni volta che si raggiunge quel punto del programma, il semaforo viene controllato: se è nello stato rosso, l'esecuzione del programma viene sospesa; se è verde, il processo continua ma il semaforo viene commutato sul rosso. Quando il secondo processo emette un'istruzione di *segnala (semaforo A)*, il semaforo è riportato al verde e il primo processo, se era in attesa, riprende l'esecuzione.

LIVELLO	NOME	OGGETTI	ESEMPI DI OPERAZIONI
13	GUSCIO	AMBIENTE DI PROGRAMMAZIONE DELL'UTENTE	ENUNCIATI NEL LINGUAGGIO DEL GUSCIO
12	PROCESSI D'UTENTE	PROCESSI D'UTENTE	SMETTI, ANNULLA, SOSPENDI, RIPRENDI
11	INDICI	INDICI	CREA, DISTRUGGI, AGGIUNGI, STACCA, CERCA, ELENCA
10	APPARECCHIATURE	APPARECCHIATURE ESTERNE COME STAMPANTI, SCHERMI VIDEO E TASTIERE	CREA, DISTRUGGI, APRI, CHIUDI, LEGGI, SCRIVI
9	SISTEMA DEGLI ARCHIVI	ARCHIVI	CREA, DISTRUGGI, APRI, CHIUDI, LEGGI, SCRIVI
8	COMUNICAZIONI	TUBI	CREA, DISTRUGGI, APRI, CHIUDI, LEGGI, SCRIVI
7	MEMORIA VIRTUALE	SEGMENTI DI MEMORIA	LEGGI, SCRIVI, RECA
6	MEMORIA SECONDARIA LOCALE	BLOCCHI DI DATI, CANALI DELLE APPARECCHIATURE	LEGGI, SCRIVI, ASSEGNA, SGOMBRA
5	PROCESSI PRIMITIVI	PROCESSI PRIMITIVI, SEMAFORI, LISTE DEI PRONTI	ASPETTA, SEGNALA, SOSPENDI, RIPRENDI
4	INTERRUZIONI	PROGRAMMI PER I GUASTI	INVOCA, MASCHERA, SMASCHERA, RITENTA
3	PROCEDURE	SEGMENTI DI PROCEDURE, PILA DI CHIAMATA, PRESENTAZIONE DELLO STATO	MARCA LA PILA, CHIAMA, RIENTRA
2	INSIEME DELLE ISTRUZIONI	PILA DI VALUTAZIONE, INTERPRETE DEL MICROPROGRAMMA, DATI SCALARI, DATI MATRICIALI	CARICA, MEMORIZZA, SALTA, SOMMA, SOTTRAI
1	CIRCUITI ELETTRONICI	REGISTRI, PORTE, LINEE ECC.	AZZERA, TRASFERISCI, ATTIVA, COMPLEMENTA

La gerarchia di astrazione è il principio organizzativo fondamentale di un sistema operativo. Ogni livello gestisce «oggetti» fisici o logici. Un programma situato a un certo livello ha accesso solo a operazioni definite ai livelli più bassi e i cui particolari interni sono nascosti. I primi sette livelli riguardano le operazioni svolte entro una sola macchina; i livelli superiori possono valersi delle risorse di più calcolatori collegati in rete.

In realtà, poiché può accadere che lo stesso semaforo controlli molti processi, il funzionamento del semaforo dev'essere un po' più complicato: esso deve comprendere un contatore e una coda di processi in attesa. A ogni istruzione *aspetta* al contatore si dà un decremento e a ogni istruzione *segnala* gli si dà un incremento. Se il valore segnato dal contatore è negativo, tutti i processi che emettono un'istruzione *aspetta* vengono inseriti nella coda d'attesa; quando viene ricevuta la successiva istruzione *segnala*, il primo processo della coda viene trasferito nella «lista dei pronti», cioè dei processi che possono già essere eseguiti. Le due operazioni definite sui semafori sono abbastanza potenti da sincronizzare i processi paralleli in svariati contesti, che vanno dalla necessità di arrestare un processo quando una memoria di transito d'ingresso è vuota o una memoria di transito d'uscita è piena alla necessità di permettere a un solo processo per volta di operare sui dati comuni a più processi.

Il sesto livello della gerarchia del sistema operativo sovraintende all'accesso ai dispositivi di memoria secondari di una singola macchina. Dai programmi di questo livello dipendono operazioni come il collocamento della testina di un'unità a dischi e la lettura di un blocco di dati. I programmi di livello superiore determinano soltanto la posizione dei dati sul disco e inseriscono la richiesta di questi dati nella coda dei compiti che il dispositivo è in attesa di sbrigare. Il processo che ha avanzato la richiesta aspetta poi a un semaforo finché il trasferimento non sia stato compiuto.

Il settimo livello è una memoria virtuale, cioè uno schema per gestire la memoria centrale e quelle secondarie del calcolatore, che dà al programmatore l'impressione che la memoria centrale sia abbastanza grande da contenere un programma e tutti i suoi dati anche se la capacità sfruttabile della memoria centrale è in realtà molto più piccola. Gli indirizzi possono essere grandi ad arbitrio e più programmi che vengano eseguiti simultaneamente possono impiegare senza conflitto gli stessi indirizzi; il sistema operativo traduce ciascun indirizzo virtuale in un indirizzo circuitale. Se un tentativo di traduzione va a vuoto perché le informazioni richieste non sono nella memoria principale, il programma gestore della memoria virtuale le ripesca automaticamente dalla memoria a disco. Può accadere che prima debba fare spazio nella memoria principale sgombrandone altri dati. Come in altre circostanze analoghe, il processo di richiesta viene interrotto finché le informazioni occorrenti non siano presenti.

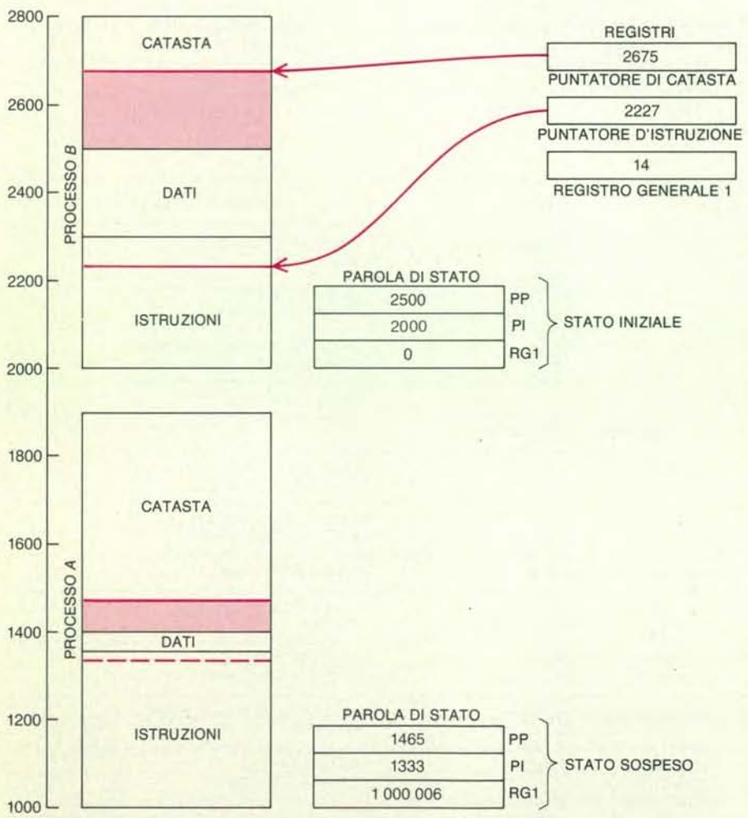
Fino al settimo livello il sistema operativo ha a che fare soltanto con le risorse di una singola macchina, mentre a partire dal livello successivo i programmi del sistema operativo abbracciano un mondo più ampio, che comprende dispositivi periferici come i terminali e le stampanti, e anche altri calcolatori collegati con la rete.

L'ottavo livello attende in modo esplicito alla comunicazione fra i processi; questo compito può essere svolto tramite un meccanismo unico, chiamato «tubo» («*pipe*»). Un tubo è un canale a senso unico nel quale un flusso di dati entra da un capo ed esce dall'altro. Il flusso ha un puntatore di scrittura, che tiene conto del numero di elementi scritti e inviati nel tubo, e un puntatore di lettura, che registra il numero di elementi letti all'altro capo. Se si richiede che vengano letti altri elementi, questa richiesta viene ritardata finché gli elementi non siano effettivamente presenti.

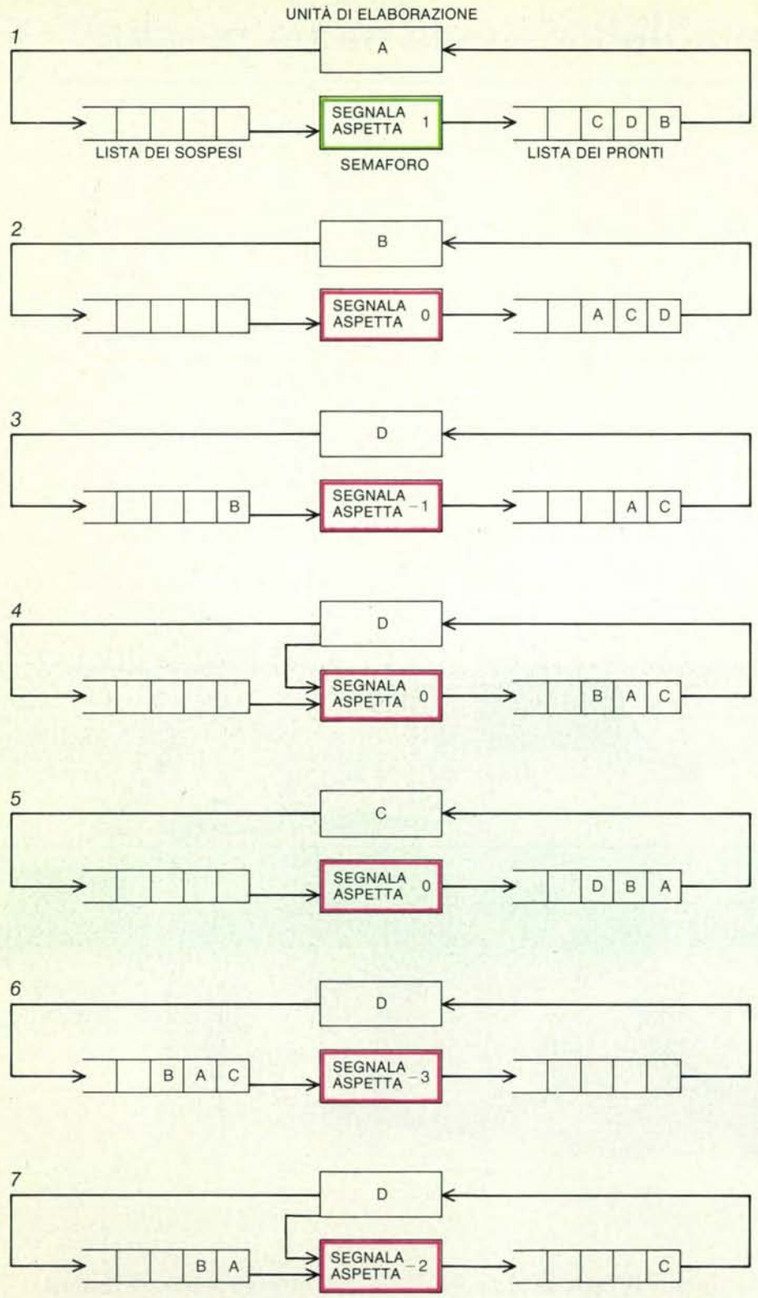
Un tubo può collegare due processi eseguiti su un'unica macchina, come quando l'uscita di un programma funge da ingresso di un altro. Ma allo stesso modo un tubo può trasmettere informazioni fra calcolatori; anzi, un insieme di tubi che colleghi i processi di tutte le macchine di una rete può servire a diffondere le «notizie» e può servire a individuare risorse in qualunque punto della rete.

Il sistema degli archivi, che s'incarica della memorizzazione a lungo termine degli archivi individuati da un nome, viene attuato al nono livello. Mentre il sesto livello si occupa della memorizzazione su disco in termini di piste e settori (che sono le divisioni, di grandezza fissa, del supporto fisico) il nono livello riguarda entità, più astratte, di lunghezza variabile, i cui termini non corrispondono necessariamente a quelli delle piste e dei settori fisici. In effetti un archivio può essere distribuito in molti settori non contigui.

Le operazioni *crea* e *distruggi* istituiscono un nuovo archivio e ne liquidano uno vecchio; *apri* e *chiudi* istituiscono e interrompono il collegamento fra un archivio e un processo. Affinché si possa esaminare il contenuto di un archivio, esso dev'essere ricopiato in un'area della memoria virtuale e affinché si possano conservare le informazioni, queste debbono essere ricopiate dalla memoria



Un processo primitivo rappresenta un programma singolo in corso di esecuzione. Qui vengono mostrati due processi primitivi caricati in un segmento della memoria principale. Il processo *B* è in corso di esecuzione. Il puntatore delle istruzioni, che è uno dei registri circuitali del calcolatore, indica l'indirizzo dell'istruzione successiva; il puntatore di catasta indica l'elemento alla sommità nella zona della memoria temporanea chiamata pila o catasta. Un calcolatore vero avrebbe un numero molto maggiore di registri di uso generale, ma qui ne è mostrato solo uno. Il processo *A* è stato sospeso, ma il contenuto di tutti i registri all'istante della sospensione è stato registrato in un'area riservata chiamata parola di stato. Il sistema operativo può commutare rapidamente da un processo all'altro. Il contenuto dei registri al momento della commutazione è collocato nella parola di stato di *B* e i registri sono ricaricati con i valori contenuti nella parola di stato di *A*.



Il semaforo dei programmi è un meccanismo per regolare i processi primitivi che debbono essere sincronizzati. Qui i processi *A*, *B* e *C* dipendono da un risultato del processo *D*. A ogni istruzione *aspetta* il semaforo fa diminuire un contatore e a ogni istruzione *segnala* lo fa aumentare. Un processo in attesa può oltrepassare il semaforo solo se il valore segnato dal contatore è maggiore di zero. All'inizio *(1)* è in corso il processo *A*; *B*, *D* e *C* sono pronti e il numero + 1 segnato dal semaforo indica che uno dei risultati di *D* è stato ottenuto. Perciò, quando emette un'istruzione *aspetta*, *A* oltrepassa subito il semaforo e si colloca nella lista dei pronti. A questo punto parte *B* *(2)*, che prima o poi emette un'istruzione *aspetta* e viene sospeso, consentendo a *D* di partire *(3)*. Quando fornisce un altro risultato, *D* emette un'istruzione *segnala*, il che permette a *B* di passare nella lista dei pronti *(4)*. *D* si colloca nella lista dei pronti e *C* parte *(5)*, ma è sospeso quando emette un'istruzione *aspetta*; allo stesso modo partono e vengono sospesi *A* e *C*, consentendo a *D* di riprendere l'esecuzione *(6)*. Quando ottiene un risultato, *D* emette un *segnala*, trasferendo *C* sulla lista dei pronti *(7)*; i cicli successivi di *D* affrancheranno *A* e *B* dalla sospensione.

virtuale in un archivio; la copiatura viene eseguita rispettivamente dalle operazioni *leggi* e *scrivi*. Se un archivio è conservato in un'altra macchina, i programmi del nono livello possono, servendosi dell'ottavo, generare un tubo che sfocia nella macchina che ospita l'archivio. (Quale sia il modo migliore di eseguire questa operazione è ancora un problema aperto.)

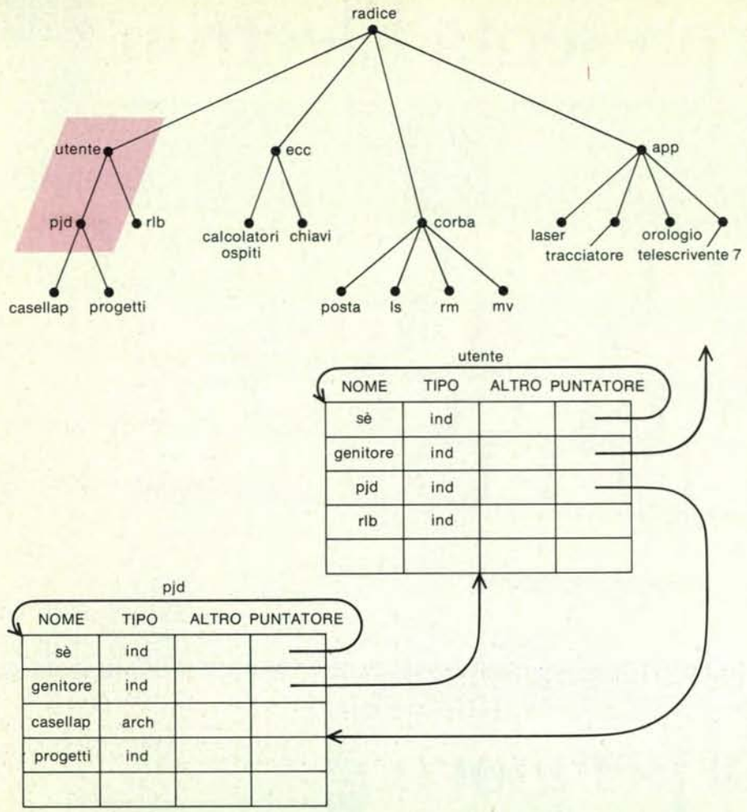
Il decimo livello procura l'accesso alle apparecchiature esterne d'ingresso e d'uscita, compresi l'orologio che fornisce l'ora, le stampanti, i tracciatori e le tastiere e gli schermi video dei terminali. Le operazioni definite su questi oggetti sono ancora una volta *crea* e *distruggi*, *apri* e *chiudi*, *leggi* e *scrivi*; e anche a questo livello si può generare un tubo per accedere a un'apparecchiatura collegata a un'altra macchina.

L'undicesimo livello gestisce una gerarchia di indici che elencano gli oggetti fisici e logici (di hardware e di software) il cui accesso dev'essere controllato: tubi, archivi, apparecchiature e anche gli indici stessi. L'elemento principale di un indice è una tabella che fa corrispondere il nome esterno di un oggetto (cioè il nome usato e fornito dall'utente, per esempio «indirizzario») a un nome interno impiegato dal sistema operativo per reperire l'oggetto. La presenza di una gerarchia è causata dalla circostanza che un indice può avere tra le sue voci anche i nomi di indici subordinati.

Ciascun indice è un elenco di voci che forniscono il nome esterno di un oggetto (memorizzato in forma di successione di caratteri), il suo nome interno (memorizzato in forma di codice binario), un indicatore del tipo al quale appartiene (tubo, apparecchiatura ecc.) e certe altre informazioni. Per esempio, di solito una voce di indice dice se dall'oggetto si può leggere, se vi si può scrivere o, se si tratta di un archivio di programmi, se può essere eseguito; inoltre ciascun genere di accesso può essere consentito a certi utenti e proibito ad altri.

Il livello degli indici è incaricato solo di registrare la corrispondenza fra i nomi esterni e interni degli oggetti; i livelli che gestiscono gli oggetti sono altri. Perciò quando si esplora un repertorio di apparecchiature per cercarvi la successione di caratteri «orologio», il risultato comunicato al programma chiamante è semplicemente il nome interno dell'orologio che segna l'ora. Questo nome interno viene poi passato a un programma del decimo livello, che effettua la lettura vera e propria dell'orologio.

Il dodicesimo livello attua i processi di utente, che sono vere e proprie macchine virtuali che eseguono programmi. È importante distinguere i processi d'utente dai processi primitivi del quinto livello. Tutte le informazioni occorrenti per definire un processo primitivo possono essere espresse nella parola di stato che inscrive il contenuto dei registri nell'unità centrale di elaborazione. Un processo d'utente comprende un processo primitivo, ma



L'albero degli archivi e degli indici organizza le risorse di un sistema di elaborazione. La radice e i nodi intermedi dell'albero sono indici che possono contenere elenchi di archivi o di indici subordinati. L'indice «corba», per esempio, contiene il codice binario dei programmi ausiliari del sistema, come i programmi per la posta elettronica e per la gestione degli archivi. Analogamente vi sono un indice «app» che elenca le apparecchiature e un indice «ecc» che contiene informazioni varie, come i calcolatori ospiti liberi e le parole chiave cifrate degli utenti. La struttura dell'indice «utente», nel quale ciascun utente del sistema tiene i propri archivi, è mostrata più in dettaglio nella parte inferiore dell'illustrazione. Ogni indice ha un puntatore verso se stesso e verso il genitore. La struttura degli indici illustrata qui è basata su quella del sistema operativo Unix.

Il processo d'utente è un calcolatore virtuale, cioè una macchina simulata che dà l'impressione di dedicarsi all'esecuzione di un singolo programma. Un processo d'utente contiene gli elementi di un processo primitivo e in più un elenco di argomenti forniti al momento dell'avvio del programma, un elenco dei porti per l'ingresso e l'uscita e una descrizione del contesto del programma. Gli argomenti sono parametri battuti dopo il nome del comando e inseriti nelle posizioni successive della matrice ARG. Tra i porti ve ne sono due impliciti per gli ingressi e le uscite, che servono quando non siano indicati altri porti. Il contesto elenca elementi come l'indice usato in quel momento.

anche molte altre cose: una memoria virtuale che contiene il programma e il suo spazio di lavoro, le informazioni fornite dall'utente quando il programma è stato avviato e un elenco di altri oggetti con cui il processo può comunicare. Un processo d'utente è molto più potente di un processo primitivo.

Il tredicesimo livello è il «guscio» («*shell*»), e lo si chiama così perché separa l'utente dal resto del sistema operativo. Esso è l'interprete di un linguaggio di comandi di livello elevato, mediante il quale l'utente dà le istruzioni al sistema. Il guscio comprende il programma ascoltatore che risponde alla tastiera del terminale; analizza ogni riga dell'ingresso per individuare nomi di programma e altre informazioni; crea e chiama un processo d'utente per ciascun programma e lo collega a tubi, archivi e apparecchiature a seconda delle necessità.

Nel sistema operativo ipotetico che stiamo descrivendo abbiamo adottato un'ipotesi importante, cioè che ingresso e uscita siano indipendenti. Le stesse operazioni fondamentali (cioè *crea, distruggi, apri, chiudi, leggi* e *scrivi*) sono definite per i tubi, gli archivi e le apparecchiature ai livelli ottavo, nono e decimo. Scrivere un blocco di dati in un archivio su disco implica una successione di eventi molto diversa da quella occorrente per trasmettere gli stessi dati a una stampante o per passarli all'ingresso di un altro programma; tuttavia né l'autore né l'utente del programma sono tenuti a preoccuparsi di queste differenze. Tutti gli enunciati *leggi* e *scrivi* del programma possono far riferimento a «porti» d'ingresso e d'uscita. I porti vengono connessi ad archivi, tubi e apparecchiature specifici solo quando il programma viene eseguito.

Questo metodo, detto del «vincolo differito», può far aumentare di molto la versatilità di un programma. Una procedura di ordinamento, per esempio, può ricevere il suo ingresso da un archivio oppure direttamente da un terminale, e quindi inviare l'uscita a un altro archivio, a un terminale o a una stampante. Senza il vincolo differito sarebbe magari necessaria una procedura distinta per ogni combinazione possibile di sorgente e destinazione.

Un altro principio che si osserva nella costruzione dei sistemi operativi porta il nome enigmatico di «occultamento dell'informazione». Ciascun livello è costruito sui livelli sottostanti, ma nasconde ai livelli soprastanti tutti i particolari interni del proprio funzionamento. Per esempio: il gestore dei processi primitivi del quinto livello procura l'illusione che tutti i processi primitivi della lista dei pronti vengano eseguiti in parallelo: i particolari delle interruzioni, delle code d'attesa e così via restano invisibili ai livelli superiori. Un programma che faccia uso di processi primitivi ha a che fare solo con un esiguo insieme di comandi esterni per creare e distruggere i processi, sospendere e riprendere la loro esecuzione, inviare e ricevere messaggi. Analogamente il

THEORIA

EDIZIONI THEORIA

In nuove edizioni integrali
riccamente annotate
i classici del pensiero scientifico

Nella collana "I Segni"

GALILEO GALILEI
SULLA LIBERTÀ DELLA SCIENZA E L'AUTORITÀ DELLE SCRITTURE
A cura di Maddalena Montinari.
Introduzione di Ehrico Bellone.
Pp. 164. L. 12.000. Ril. in tela L. 16.000 (N. 1)

RENÉ DESCARTES
IL MONDO OVVERO
TRATTATO DELLA LUCE E L'UOMO
Introduzione e cura di Maurizio Mamiani.
Pp. 198. L. 16.000. Ril. in tela L. 20.000 (N. 2)

LUIGI GALVANI
MEMORIE SULL'ELETTRICITÀ ANIMALE AL CELEBRE ABATE LAZZARO SPALLANZANI
Introduzione e cura di Mauro Recenti.
Con un saggio di Maurizio Mamiani.
Pp. 184. L. 16.000. Ril. in tela L. 20.000 (N. 3)

ISAAC NEWTON
IL SISTEMA DEL MONDO E GLI SCOLII CLASSICI
Introduzione e cura di Paolo Casini.
Pp. 167. L. 25.000. Ril. in tela L. 30.000 (N. 4)

NICOLA COPERNICO
COMMENTARIOLUS
A cura di Edward Rosen.
Introduzione di Francesco Barone.
Pp. 108. L. 16.000. Ril. in tela L. 20.000 (N. 5)

JOHANNES KEPLER
SOMNIUM
A cura di Edward Rosen.
Introduzione di Giovanni Godoli.
Pp. 224. L. 35.000. Ril. in tela L. 40.000 (N. 6)

Se non li trovate in libreria, incollate su cartolina postale il coupon e spedite a:
THEORIA, Via Domodossola 11, 00183 Roma

☐ Contrassegno (spese post. L. 1.500)
☐ Allegato assegno non trasferibile
Desidero ricevere i seguenti volumi

1 2 3 4 5 6

NOME

INDIRIZZO

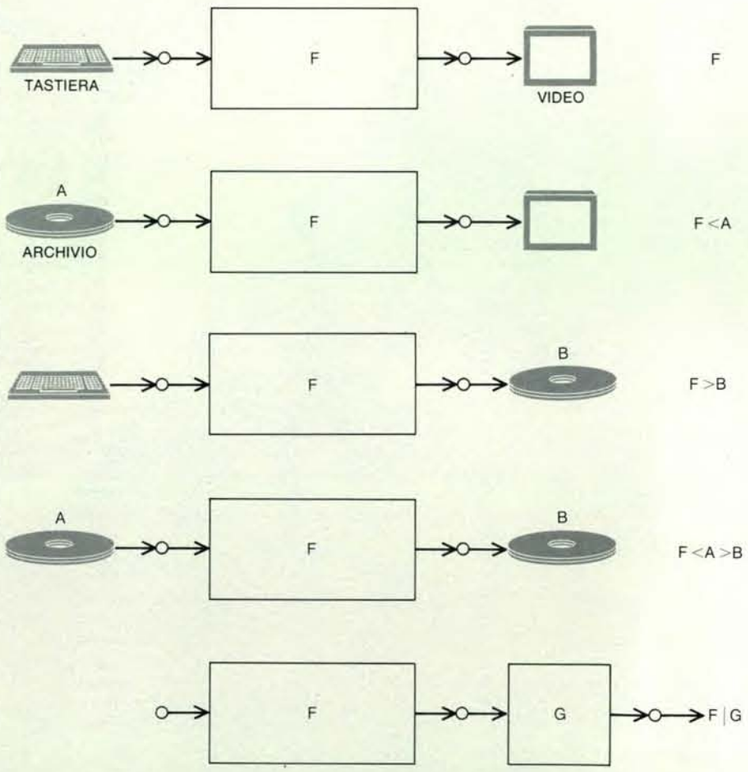
gestore dei processi d'utente del dodicesimo livello procura l'illusione che ciascun programma venga eseguito nella sua macchina; la creazione del processo primitivo, lo spazio di lavoro e i collegamenti con i porti d'ingresso e d'uscita sono tutti nascosti.

Come nessun livello ha adito al funzionamento interno dei livelli inferiori, allo stesso modo nessun livello dipende da ipotesi relative ai livelli superiori. Il gestore della memoria virtuale (settimo livello) deve avere accesso al sistema delle interruzioni (quarto livello) e al sistema delle memorie secondarie (sesto livello), ma non sa nulla sulle strutture degli archivi (nono livello). La stratificazione del sistema operativo agevola la sua costruzione, poiché i livelli possono essere installati e verificati uno alla volta dal basso verso l'alto.

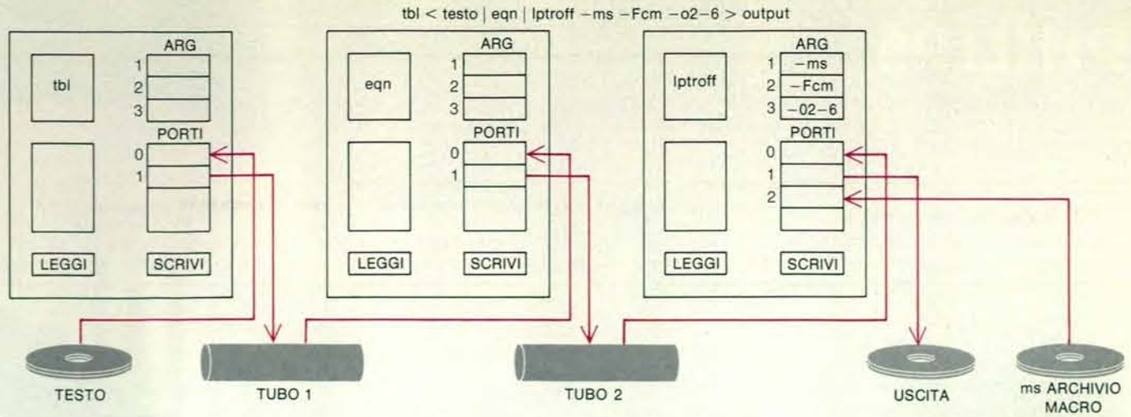
La descrizione del sistema operativo che è stata data fin qui è statica: è stato fatto l'elenco delle sue parti, ma non si è visto come funzionino. Lo stato del sistema muta a mano a mano che i comandi vengono eseguiti. Gli esempi di funzionamento che ora daremo sono basati sul sistema operativo Unix, che possiede molte delle caratteristiche del sistema ipotetico che abbiamo illustrato sopra.

L'utente vede il calcolatore come un grande sistema dotato di molte risorse utili. Alcune di queste risorse sono programmi memorizzati in codice binario che possono essere eseguiti semplicemente dando un nome d'archivio. In un sistema Unix questa classe potrà comprendere il programma della data, i compilatori per i linguaggi di alto livello e i programmi per allestire documenti, inclusi i programmi per compilare nei formati tipo tabelle, equazioni e testi ordinari. Altri elementi del sistema sono archivi di dati, che magari custodiscono documenti di vario genere, compreso il «codice sorgente» dei programmi. Sono accessibili anche talune apparecchiature fisiche, come i terminali, l'orologio e le stampanti.

Gli indici che elencano tutte queste risorse sono disposti secondo un albero inverso, con l'indice di livello più alto alla radice. Alcuni indici sono riservati al sistema; tra questi potranno essere elenchi di apparecchiature, di comandi e di archivi contenenti dati di vario tipo, come per esempio le parole d'ordine degli utenti autorizzati. Un indice intitolato «utente» contiene un sottoindice per ciascuna persona accreditata presso il sistema e a sua volta ciascun utente può costruirsi un proprio albero di sottoindici, in cui memorizzare tutti gli archivi, i tubi e i dispositivi da lui creati.



Una componente, o parte, del software è un programma con un unico flusso d'ingresso e un unico flusso d'uscita; è una struttura che serve per combinare in vari modi programmi e apparecchiature. Se non sono specificate un'altra sorgente e un'altra destinazione, un programma viene collegato implicitamente alla tastiera e allo schermo video dell'utente. Il segno «<» designa una sorgente d'ingresso e il segno «>» una destinazione di uscita. Il segno «|» istituisce un «tubo», un canale a senso unico che può collegare l'uscita di un programma con l'ingresso di un altro programma.



La catena del processo mette insieme tre programmi, due tubi e tre archivi per la preparazione di un testo da stampare. Il primo programma, *tbl*, prende in ingresso un archivio chiamato «testo» e immette comandi che dànno il formato a materiale da tabulare. L'uscita di *tbl* è convogliata a *eqn*, che compie operazioni analoghe di formato su equazioni. Il secondo tubo porta il testo a *lptroff* (contrazione di «laser printer typesetter runoff») che completa le operazioni di formato. Come argomenti della riga di comando sono assegnate tre scelte per il programma *lptroff*: *−ms* dà al programma l'istruzione di aprire un archivio «macro» detto «ms» per espandere i codici di formato abbreviati incontrati nel testo; *−Fcm* specifica una fonte di caratteri chiamata Computer Modern e *− o2−6* indica che debbono essere prodotte solo le pagine da 2 a 6 dell'uscita. L'uscita è poi inviata a un archivio per essere stampata.

Gli utenti del sistema per lo più non scrivono programmi nuovi, ma ne impiegano di già esistenti; questa abitudine viene incoraggiata dal modo in cui è concepito il sistema operativo, in particolare dal principio dell'indipendenza fra ingresso e uscita. La maggioranza dei programmi contenuti nelle biblioteche dei sistemi sono «pezzi di software» che possono operare, in maniera intercambiabile, con varie sorgenti di ingresso e varie destinazioni di uscita.

Quando un utente s'inserisce, e in genere lo fa dando la sua parola d'ordine, il sistema operativo crea un processo d'utente che comprende una copia del programma guscio. L'ingresso del guscio è collegato alla tastiera del terminale dell'utente e la sua uscita è collegata allo schermo video dello stesso terminale. Il guscio «ascolta», senza intraprendere alcuna azione, finché non sia stata battuta un riga completa dell'ingresso, il che viene segnalato dalla ricezione del carattere «ritorno carrello». A questo punto la riga viene esplorata per estrarne i nomi dei programmi che vengono chiamati e i valori da fornire come argomenti di quei programmi. Per ciascun programma chiamato a questo modo, il guscio crea un processo d'utente, compresi una copia del codice esecutivo del programma e uno spazio di lavoro. I processi sono collegati in conformità con il flusso di dati contenuto nella riga di comando d'ingresso.

Con il linguaggio di comando del guscio dell'Unix si possono descrivere operazioni di complessità ragguardevole. Per esempio, una successione di operazioni che dà il formato a un archivio chiamato «testo» può essere avviata da questa riga di comandi:

tbl < testo | eqn | *lptroff* > *output*

Qui il primo programma chiamato è *tbl*: la sua funzione è quella di esplorare un archivio per trovarvi le descrizioni delle tabelle di informazioni e inserirvi i comandi relativi al formato. Il simbolo «<» indica che *tbl* deve prendere il suo ingresso dall'archivio «testo». L'uscita di *tbl* è convogliata da un tubo (il simbolo «|») all'ingresso di *eqn*, che fornisce i comandi relativi al formato di equazioni, qualunque sia la loro descrizione. Un tubo manda poi l'uscita di *eqn* al programma *lptroff*, un altro programma di formato che prepara il resto del testo per la stampa (il suo nome è una contrazione di *«laser printer typesetter runoff»*, «composizione e stampa a laser»). Infine, il simbolo «>» indica che, ricevuto il suo formato, il documento viene scritto in un archivio chiamato «output». Esso è pronto per essere inviato a una stampante laser di elevata qualità che lavora in modo simile a una fotocopiatrice.

Se queste operazioni di formato e stampa di documenti debbono essere compiute spesso, battere un comando così complicato diventa ben presto noioso. L'Unix incoraggia l'utente a memorizzare i comandi complicati in archivi esecutivi chiamati «scritture di guscio», trasformandoli in comandi più semplici. Si potrebbe creare un archivio chiamato *sl* con questo contenuto:

tbl < $1 | eqn | *lptroff* > $2 .

I nomi degli archivi d'ingresso e d'uscita qui sono stati sostituiti dalle variabili $1 e $2. Quando si invia il comando *sl*, le variabili vengono sostituite a loro volta dagli argomenti che fanno seguito al comando. Per esempio se si batte

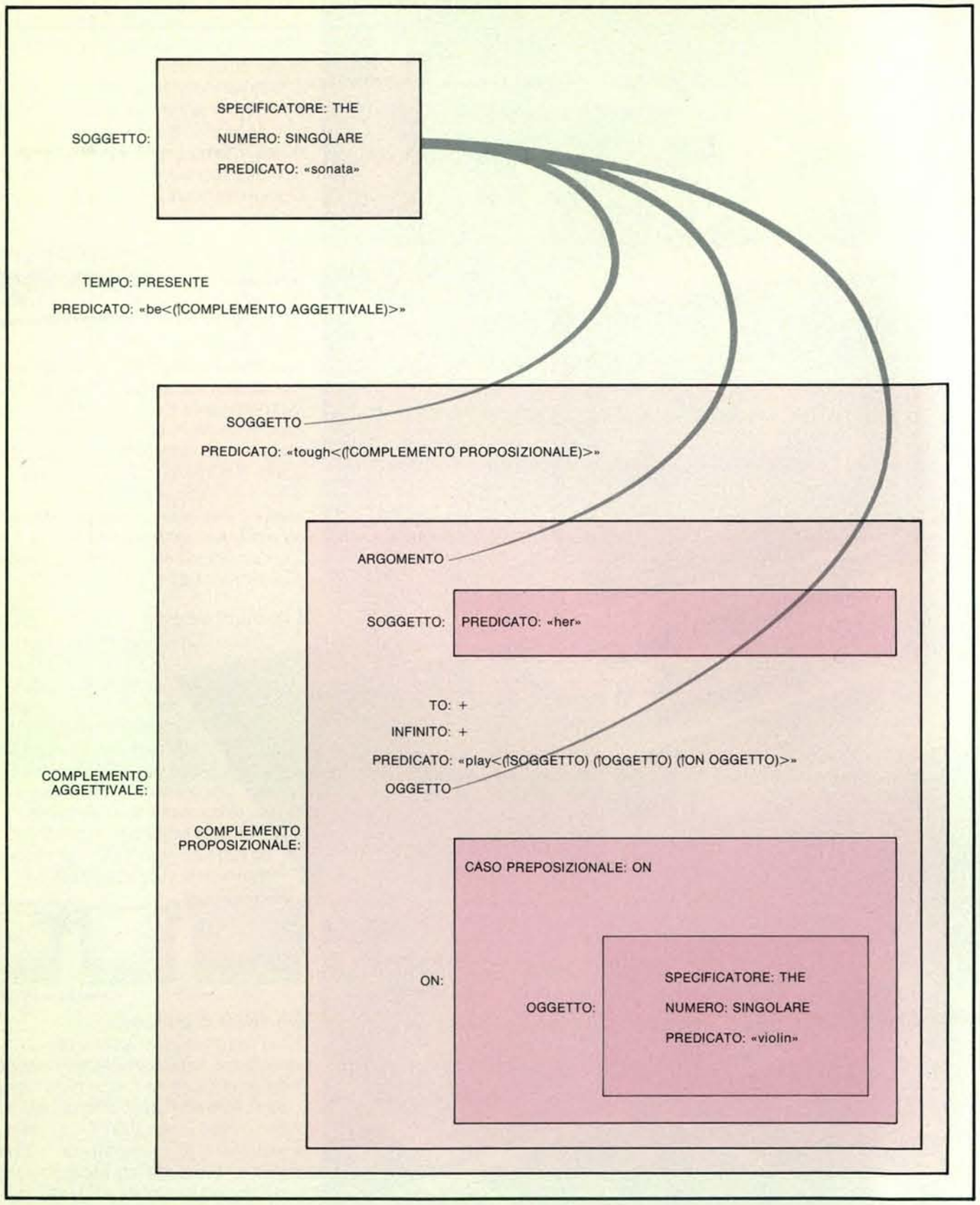
lp testo *output*

$1 viene sostituita da «testo» e $2 da «output» e si ottiene lo stesso effetto che con il comando più lungo descritto sopra.

Dobbiamo descrivere ancora una parte, piccola ma essenziale, del sistema operativo. Dal momento che le varie componenti del sistema operativo hanno il compito di caricare tutti i programmi nella macchina, ci si può naturalmente domandare come venga caricato e avviato il sistema operativo stesso. La risposta è: con una «sequenza di lancio» *(bootstrap sequence)*. La sequenza comincia con un programma di due sole istruzioni iscritte in modo permanente in una memoria di sola lettura; le istruzioni sono quindi presenti anche quando la macchina è spenta. Questo programmino comincia a caricare da un disco un programma un po' più grande, che poi assume il controllo e carica a sua volta il sistema operativo.

Il principio gerarchico che abbiamo visto applicato qui all'organizzazione di un sistema operativo si è dimostrato di grande utilità in tutto il dominio delle scienze naturali. In ultima analisi, le strutture e gli eventi del mondo naturale abbracciano molti ordini di grandezza nello spazio e nel tempo e non possono essere colti tutti in una volta: non è possibile comprendere l'evoluzione di una galassia tracciando le traiettorie degli atomi che la compongono. Fra tutti gli oggetti costruiti dall'uomo è nei calcolatori che si osserva la massima disparità tra i componenti più piccoli e quelli più grandi. I progettisti dei sistemi operativi hanno cominciato ad affrontare questa gamma così ampia di scale creando una gerarchia di astrazione crescente.

The sonata is tough for her to play on the violin.



La rappresentazione di una frase in un modo che renda esplicite le relazioni linguistiche fra le sue parti è da sempre un obiettivo della linguistica ed è anche un aspetto inevitabile della ricerca di un software che «comprenda» il linguaggio, o almeno possa trarre inferenze da dati linguistici. Qui è stata data una frase in forma di «struttura funzionale». In questo tipo di rappresentazione, quando una parte di una frase svolge un ruolo in un'altra parte, è «annidata» in quest'ultima. Qui l'annidamento è visualizzato dall'inserimento di un riquadro in un altro oppure (in tre casi) da una linea di collegamento. La frase è stata analizzata da Ronald M. Kaplan e Joan Bresnan della Stanford University e del Palo Alto Research Center della Xerox Corporation. Un altro diagramma di struttura funzionale è presentato nell'illustrazione delle pagine 82 e 83.

Software per lavorare con il linguaggio

Si possono manipolare facilmente i simboli linguistici, come nel software per l'elaborazione di testi, ma ogni tentativo di trattare il significato con mezzi automatici è stato ostacolato dall'ambiguità delle lingue umane

di Terry Winograd

Nella mitologia popolare il calcolatore è una macchina per la matematica: è stato progettato per effettuare calcoli numerici. Ma in realtà è una macchina per il linguaggio: la sua potenza sta fondamentalmente nella capacità di manipolare oggetti linguistici, simboli a cui è stato assegnato un significato. In effetti, il «linguaggio naturale» (il linguaggio che parliamo e scriviamo, in contrapposizione ai linguaggi «artificiali» in cui sono scritti i programmi per i calcolatori) occupa un posto centrale nella scienza del calcolatore. Molte fra le prime ricerche in questo settore erano rivolte alla decrittazione di codici militari e negli anni cinquanta i tentativi di far tradurre al calcolatore testi da una lingua naturale in un'altra portarono progressi cruciali, anche se l'obiettivo in sé non fu raggiunto. La ricerca oggi si è spostata sul progetto, ancor più ambizioso, di rendere il linguaggio naturale un mezzo di comunicazione con i calcolatori.

Oggi i ricercatori stanno sviluppando teorie unificate della computazione che abbracciano sia le lingue naturali sia i linguaggi artificiali. Qui concentrerò la mia attenzione sulle prime, cioè sul linguaggio della comunicazione umana di tutti i giorni. Il campo è vasto, e il software pertinente molto, in parte costituito da programmi di larga diffusione e di grande successo commerciale. Case, uffici e scuole sono stati invasi da un esercito di microcalcolatori, per lo più usati, almeno in parte, per il *word processing*, cioè per l'elaborazione di testi scritti. Altre applicazioni sono ancora allo stadio ipotetico, ben lontano dalla realizzazione. La fantascienza è popolata di robot che chiacchierano come esseri umani, solo con una leggera sfumatura metallica nella voce. I tentativi reali di far conversare i calcolatori sono incorsi in grandi difficoltà, e i migliori prototipi di laboratorio presentano solamente una pallida somiglianza con la competenza linguistica caratteristica del bambino medio.

La gamma del software per l'elaborazione del linguaggio è tanto vasta da rendere impossibile una rassegna esaustiva: mi concentrerò su quattro tipi di programmi, e cioè su quelli relativi alla traduzione automatica, all'elaborazione di testi, alla risposta a interrogazioni, e ai cosiddetti sistemi di coordinamento, complementari alla posta elettronica. In ogni caso l'elemento chiave per stabilire ciò che è possibile sta nell'analisi della natura della competenza linguistica e del modo in cui questa competenza è in rapporto con le strutture di regole formali che costituiscono la base teorica per tutto il software.

L'idea che un calcolatore potesse tradurre testi nacque molto prima che queste macchine venissero prodotte commercialmente. Nel 1949, quando i pochi calcolatori funzionanti si trovavano tutti nei laboratori militari, il matematico Warren Weaver, uno dei pionieri della teoria dell'informazione, mise in evidenza come le tecniche sviluppate per la decrittazione potessero essere applicate alla traduzione automatica.

A tutta prima il compito sembra immediato. Data una frase in un linguaggio sorgente, due operazioni fondamentali producono la frase corrispondente nel linguaggio obiettivo. In primo luogo si sostituiscono le singole parole con le corrispondenti traduzioni; poi le parole tradotte vengono riordinate e messe a punto nei particolari. Prendiamo la traduzione dell'inglese *«Did you see a white cow?»* nello spagnolo *«¿Viste una vaca blanca?»*. Prima di tutto si devono conoscere le corrispondenze fra i singoli vocaboli: *vaca* per *cow* e via dicendo. Le parole *did* e *you* non sono tradotte direttamente, ma espresse attraverso la forma del verbo *viste*. L'aggettivo *blanca* segue il nome, anziché precederlo come in inglese. Infine *una* e *blanca* sono nella forma femminile corrispondente a *vaca*. Gran parte dei primi studi sulla traduzione automatica si sono fermati al problema tecnico di inserire un esteso dizionario nella memoria di un calcolatore e di dotare la macchina di procedimenti di ricerca efficienti. Nel contempo il software per il trattamento della grammatica era basato sulle teorie della struttura del linguaggio allora più diffuse, affiancate da una serie di regole empiriche.

Quei programmi davano traduzioni tanto cattive da risultare incomprensibili. Il problema è che il linguaggio naturale non incorpora il significato nello stesso modo in cui un codice cifrato incorpora un messaggio. Il significato di una frase in una lingua naturale dipende non soltanto dalla forma della frase stessa, ma anche dal contesto. Lo si può vedere bene prendendo in considerazione alcuni esempi di ambiguità.

La forma più semplice di ambiguità è quella lessicale: una singola parola può avere più significati possibili. Un caso tipico è quello del sostantivo *bank* in inglese, che può significare sia banca, sia banchina, riva. *«Stay away from the bank»* può essere sia un consiglio per un piccolo risparmiatore, sia un avvertimento per un bambino troppo vicino a un fiume. Traducendo la frase in spagnolo, si deve scegliere fra *banco* e *orilla* e nulla, nella frase presa in se stessa, ci dice quale sia il significato inteso. Nel tentativo di affrontare l'ambiguità lessicale nel software per la traduzione, si sono presi in considerazione da un lato l'inserimento nel testo tradotto di tutte le possibilità e dall'altro l'analisi statistica del testo sorgente per decidere quale sia la traduzione appropriata. Per esempio, è probabile che la scelta corretta sia *orilla*, se nel testo sorgente si trovano, vicino alla frase in questione, parole che

si riferiscono a fiumi e acqua. La prima strategia produce testi complessi e illeggibili; la seconda dà la scelta corretta in molti casi, ma in molti altri produce una scelta sbagliata.

Nell'ambiguità strutturale il problema va al di là della singola parola. Si consideri una frase come «*He saw that gasoline can explode*». Essa ha due interpretazioni («Ha visto esplodere questo bidone di benzina» e «Ha visto che la benzina può esplodere»), basate su usi molto diversi delle parole *that* e *can*. Pertanto la frase ha due strutture grammaticali possibili, e il traduttore deve operare una scelta. [Una situazione analoga si ha nella frase italiana «Una vecchia porta la sbarra», dove «porta» può essere interpretato come sostantivo o come verbo e «la» come pronome o come articolo.] *(Si veda l'illustrazione in basso a pagina 75.)*

Un'ambiguità di «struttura profonda» è ancora più sottile: due letture di una frase possono avere la stessa struttura grammaticale apparente, ma differire, ciononostante, in significato. «*The chickens are ready to eat*» [in italiano, per conservare l'ambiguità, potremmo tradurre: «I polli sono pronti per il pranzo»] implica che qualcosa sta per mangiare qualcosa, ma da che parte stanno i polli? Uno dei grandi progressi nella linguistica teorica dagli anni cinquanta in poi è stato lo sviluppo di un formalismo in cui è possibile rappresentare la struttura profonda del linguaggio, ma il formalismo non è di molto aiuto nel dedurre la struttura profonda voluta di una particolare frase.

Un quarto tipo di ambiguità, l'ambiguità semantica, si ha quando un sintagma può svolgere diversi ruoli nel significato complessivo di una frase. La frase «Giovanni vuole sposare una norvegese» ci offre un esempio. In un significato della frase, il sintagma «una norvegese» ha valore referenziale: Giovanni intende sposare una persona ben precisa e chi ha pronunciato la frase ha scelto, per descriverla, un suo attributo, il fatto di esser nata in Norvegia. In un altro significato della frase, invece, il sintagma ha valore attributivo. Né Giovanni né chi ha pronunciato la frase hanno in mente una particolare persona: la frase vuol dire semplicemente che Giovanni spera di sposare una qualche fanciulla di nazionalità norvegese.

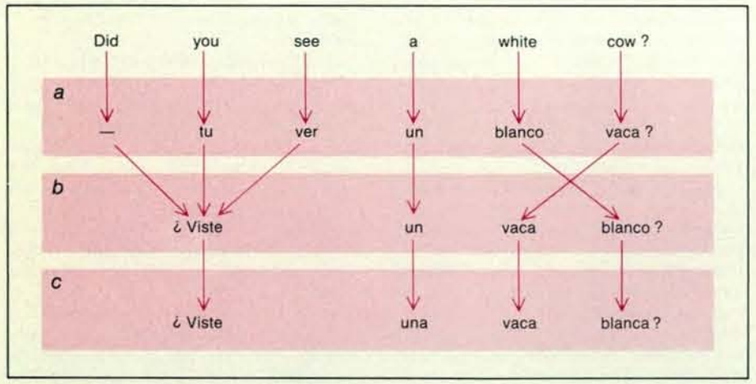
Un quinto tipo di ambiguità può essere definito «ambiguità pragmatica» e deriva dall'uso di pronomi e nomi particolari come, in inglese, *one* e *another*. Prendiamo per esempio la frase «*When a bright moon ends a dark day, a brighter one will follow*»: che cosa seguirà, un giorno più luminoso o una luna più luminosa? A volte il software può limitarsi a tradurre il pronome o il nome ambigui, conservando l'ambiguità nella traduzione, ma in molti casi questa strategia non può essere seguita. Per questa frase, per esempio, nella traduzione in italiano si deve optare per «un altro» o «un'altra», togliendo necessariamente l'ambiguità. In una traduzione spagnola della frase «*She dropped the plate on the table and broke it*», per rendere *it* si deve scegliere fra il maschile «lo» e il femminile «la»: la scelta forza il traduttore a decidere se ciò che è stato rotto era il *plato* (maschile) o la *mesa* (femminile). [Traducendo in italiano, si può conservare l'ambiguità: «*Lasciò cadere il piatto sul tavolo e lo ruppe*».]

In molte frasi ambigue il significato è ovvio per un lettore umano, ma solo perché egli può fare entrare in gioco la sua comprensione del contesto. Così «L'insalata è pronta per la cena» è una frase priva di ambiguità perché sappiamo che l'insalata è un essere inanimato. «C'è un uomo nella stanza con un cappello verde» non è ambigua perché sappiamo che le stanze non portano cappelli. Senza conoscenze di questo tipo praticamente qualunque frase sarebbe ambigua.

Una traduzione meccanica completamente automatica e di elevata qualità non è possibile, ma disponiamo di software che può facilitare la traduzione. Un esempio è la meccanizzazione di strumenti di traduzione come dizionari e frasari. Si va da sistemi complessi rivolti ai traduttori tecnici, in cui la funzione della «consultazione di un lemma» è parte di un programma di elaborazione di testi in più lingue, ad archivi di frasi fatte, per uso dei turisti, memorizzate nei calcolatori tascabili. Un'altra strategia è quella di elaborare il testo manualmente per renderlo accessibile a una traduzione meccanica. Una persona che funge da «preredattore» prende un testo nel linguaggio sorgente e ne crea un secondo, sempre nel linguaggio sorgente, semplificato nel modo più opportuno per facilitare la traduzione a opera della macchina. Si possono eliminare le parole con più significati, insieme con le costruzioni grammaticali che complicano l'analisi sintattica. Si possono sopprimere le congiunzioni che creano ambiguità, oppure si può risolvere l'ambiguità inserendo speciali forme di punteggiatura, come per l'espressione «*the (old men) and (women)*», dove le parentesi specificano che l'aggettivo *old* si riferisce solo a *men* e non anche a *women*. Dopo che la macchina ha effettuato la traduzione, un «postredattore» può controllare gli svarioni e limare il testo tradotto.

A volte lo sforzo può essere economicamente conveniente. Innanzitutto, il preredattore e il postredattore non debbono essere necessariamente bilingui, come invece dovrebbe essere un traduttore. Inoltre, se il testo di partenza (caso tipico quello di un manuale d'istruzioni) deve essere tradotto in varie lingue, un forte investimento nella preredazione può essere giustificato dalla sua utilità per tutte le traduzioni. Se l'autore del testo può poi imparare la forma meno ambigua del linguaggio sorgente, non è necessario il preredattore. Infine, il software può servire nel controllo del testo preredatto, per verificare che soddisfi i requisiti per l'immissione nel sistema di traduzione (anche se questo non costituisce una garanzia di una traduzione accettabile in uscita).

Un sistema di traduzione automatica che usa pre- e postredazione è in servizio dal 1980 alla Pan-American Health Organization, dove ha già tradotto più di un milione di parole di testo dallo spagnolo all'inglese. Un nuovo sistema è in fase di sviluppo per la Comunità economica europea e ha come obiettivo la traduzione di documenti fra le lingue ufficiali della comunità: danese, francese, greco, inglese, italiano, olandese e tedesco. Le ricerche teoriche sulla sintassi e sul significato continuano, ma non ci sono stati passi in avanti nella traduzione automatica.



Negli anni cinquanta si pensava che la traduzione automatica di un testo da una lingua in un'altra fosse un'impresa realizzabile. Nella prima fase del procedimento *(a)* il calcolatore dovrebbe consultare un dizionario bilingue per identificare la traduzione delle singole parole (in questo caso gli equivalenti spagnoli delle parole della frase «*Did you see a white cow?*»). Poi le parole tradotte verrebbero riordinate secondo la grammatica della lingua d'arrivo *(b)*. Cambiamenti in questa fase potrebbero comportare l'eliminazione o l'aggiunta di parole. Infine *(c)* verrebbe messa a punto la morfologia della traduzione (per esempio per rispettare la concordanza di genere).

L'ambiguità che pervade il linguaggio naturale continua a limitare le possibilità, per motivi che esaminerò meglio più avanti.

Passiamo quindi all'elaborazione di testi o *word processing*, cioè al software che funge da ausilio nella preparazione, nell'impaginazione e nella stampa dei testi. Gli elaboratori di testi affrontano soltanto la manipolazione e la visualizzazione di stringhe di caratteri e, pertanto, riguardano solo aspetti superficiali della struttura del linguaggio. Ciononostante, pongono problemi tecnici fondamentali alla progettazione del software. In alcuni casi il prodotto finale di un programma di elaborazione di testi non è altro che una successione di righe di testo; in altri è una disposizione complessa di elementi tipografici, a volte intercalati con disegni. In altri casi ancora è un documento strutturato, con intestazioni di capitoli, numeri di sezione e via dicendo, con un sommario e un indice analitico compilati dal programma stesso.

I problemi chiave nella costruzione di programmi per l'elaborazione di testi riguardano gli aspetti della rappresentazione e dell'interazione. La rappresentazione è il compito di formulare strutture di dati che possano essere manipolate comodamente dal software ma che in aggiunta provvedano a ciò che preme effettivamente all'utente del sistema, per esempio la disposizione della pagina stampata. Il problema dell'interazione è il problema del modo in cui l'utente esprime le sue istruzioni e del modo in cui il sistema risponde a esse.

Si consideri il problema fondamentale dell'uso di dispositivi di memoria di un calcolatore per conservare una successione codificata di caratteri del linguaggio naturale. I primi dispositivi in cui il testo veniva codificato erano punzonatrici di schede e telescriventi e, di conseguenza, i primi schemi di codificazione dei testi furono progettati su misura per questi dispositivi. La telescrivente è sostanzialmente una macchina per scrivere che trasforma la pressione dei tasti in codici numerici che possono essere trasmessi elettronicamente. Di conseguenza, esistono codici di telescrivente per la maggior parte dei tasti di una macchina per scrivere; essi includono codici per i caratteri alfabetici dalla *A* alla *Z*, per le cifre da 0 a 9 e per i comuni segni di interpunzione come il punto e la virgola. Più difficile è stabilire degli standard per simboli come #, @, ¢ e }. E che cosa dire di tasti che non stampano alcunché, come il tasto di tabulazione, il tasto di ritorno del carrello e il tasto che sposta indietro di uno spazio la testina di scrittura?

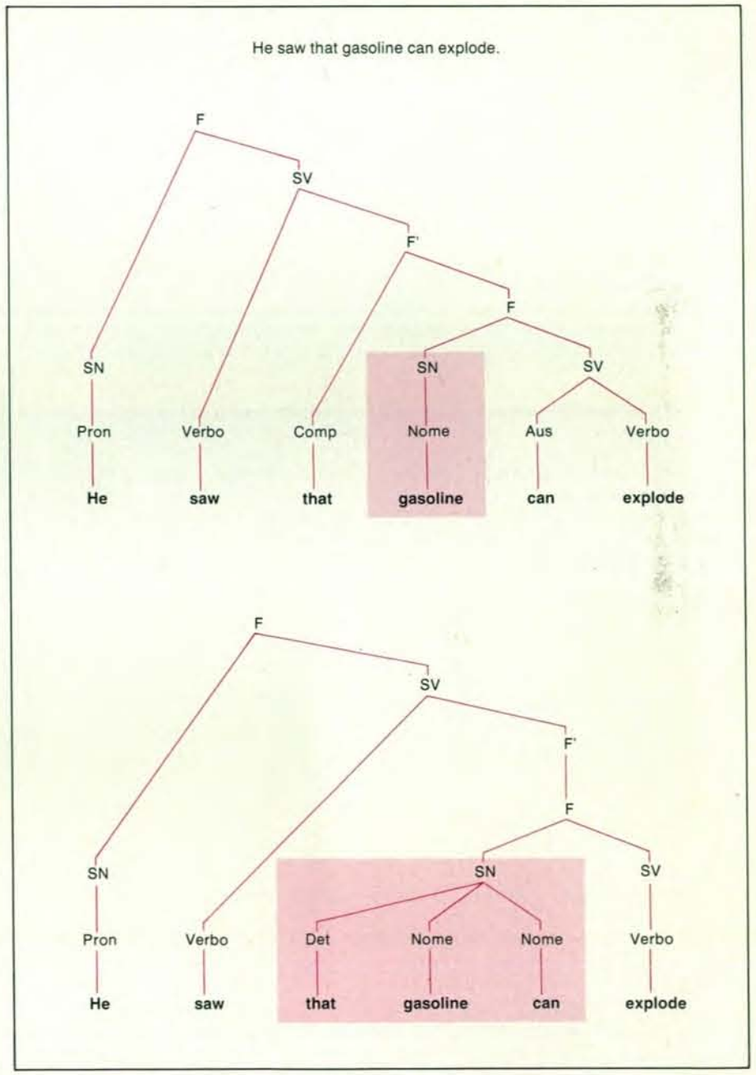
Una particolarità della codificazione dei testi può servire bene come esemplificazione delle difficoltà che si incontrano nello scegliere uno standard. Il codice di telescrivente distingue fra un ritorno del carrello (che riporta il carrello all'inizio della riga, senza far avanzare la carta) e un comando di avanzamento della carta *(line feed)*, che fa avanzare la carta senza

Stai lontano dal **cane**.

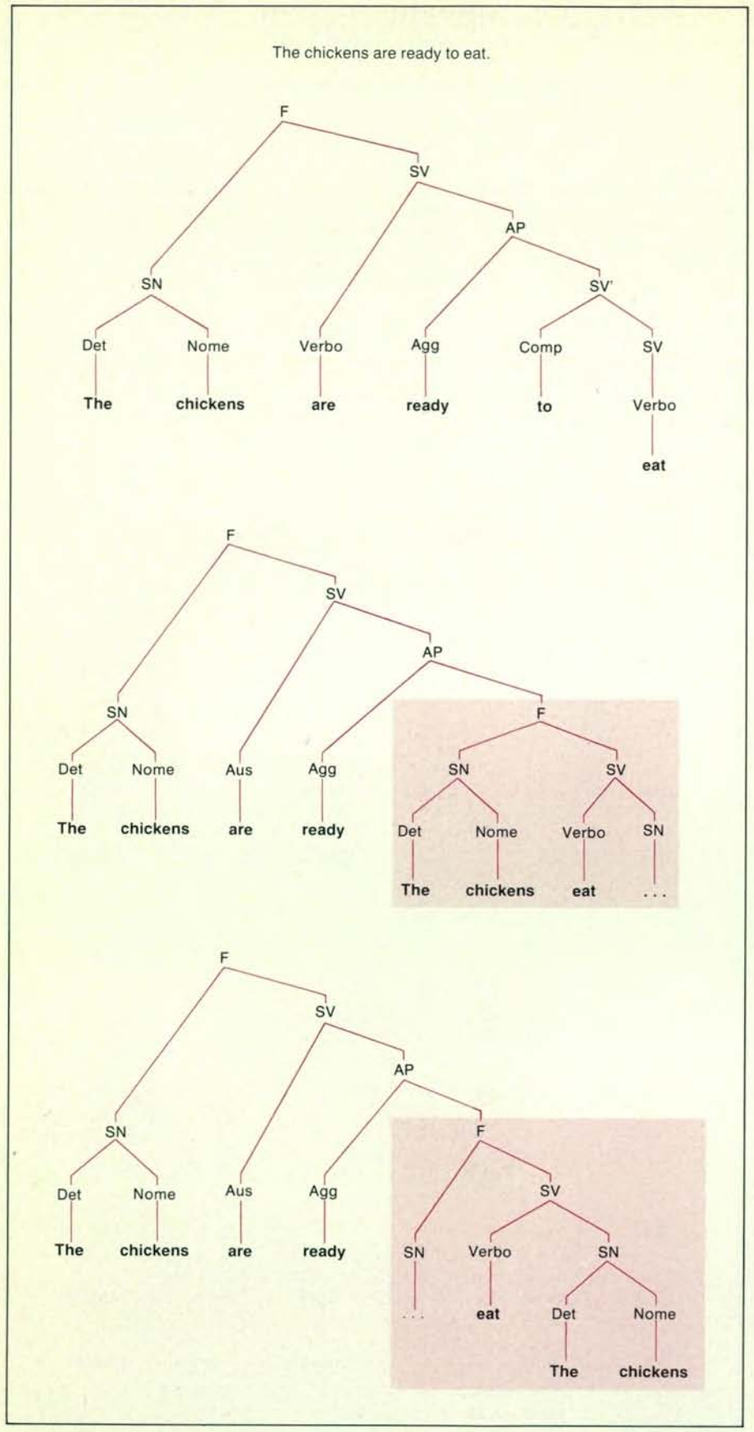
cane *n.* 1. Mammifero domestico, carnivoro, fedele amico dell'uomo.

cane *n.* 2. Parte del meccanismo di un'arma da fuoco portatile che, scattando, provoca l'accensione della carica di lancio.

L'ambiguità di significato è onnipresente nelle lingue naturali (cioè nelle lingue che parliamo e scriviamo) e crea gravi difficoltà al tentativo di far tradurre a un calcolatore testi da una lingua a un'altra. Qui abbiamo un esempio del tipo più semplice, l'ambiguità lessicale, in cui una parola in una frase ha più di un significato possibile. In questo caso la parola è «cane» *(in colore)* che può riferirsi o all'animale o a quella parte di una rivoltella che determina lo sparo. Un traduttore deve scegliere. Le successive quattro illustrazioni mostrano tipi più complessi di ambiguità.



Si ha ambiguità strutturale quando una frase può essere descritta da più strutture grammaticali. Qui le possibilità alternative per la frase «*He saw that gasoline can explode*» sono visualizzate in forma di alberi grammaticali. In uno di questi alberi la frase possiede una proposizione subordinata il cui soggetto è *gasoline (in colore)*; la frase esprime allora il riconoscimento di una proprietà di questo materiale («Ha visto che la benzina può esplodere»). Nell'altro albero *gasoline can* fa parte di un sintagma nominale *(SN)* e significa «bidone di benzina»; la frase allora dice che una certa persona ha assistito a una specifica esplosione («Ha visto esplodere quel bidone di benzina»).



Si ha ambiguità di struttura profonda quando una frase ha un'unica struttura superficiale ma, ciononostante, ha più significati possibili. In questo esempio la frase è *«The chickens are ready to eat»*. La sua struttura grammaticale *(in alto)* lascia ambiguo il ruolo dei polli: secondo un'interpretazione i polli mangiano («I polli sono pronti per mangiare»), seconda l'altra vengono mangiati («I polli sono pronti per essere mangiati»). Gli alberi di struttura profonda rendono esplicito il ruolo dei polli: sono il soggetto della frase *(al centro)* e il loro cibo è indeterminato, oppure sono il complemento oggetto *(in basso)* e restano indeterminati i soggetti che li mangeranno.

modificare la posizione del carrello. Di conseguenza la fine di una riga è contrassegnata da due caratteri: un ritorno del carrello e un avanzamento della carta. Un solo codice sarebbe sufficiente e così alcuni programmi eliminano il ritorno del carrello o l'avanzamento della carta, oppure li sostituiscono ambedue con un terzo codice, ancora diverso. Il problema è che programmi diversi utilizzano convenzioni diverse, e pertanto le righe codificate da un programma possono non essere leggibili da un altro.

Il problema si aggrava quando si prende in considerazione un repertorio completo di caratteri (segni di interpunzione, simboli matematici, segni diacritici come l'*Umlaut*). Inoltre, oggi l'elaborazione di testi al calcolatore si va estendendo a lingue come il cinese e il giapponese, che richiedono migliaia di caratteri ideografici, e a lingue come l'arabo e l'ebraico, che sono scritte da destra verso sinistra. Gli schemi di codificazione adatti per l'inglese o per l'italiano sono del tutto privi di utilità per alfabeti con migliaia di caratteri. Va detto poi che gli schemi continuano a variare perché nella progettazione dei sistemi di elaborazione svolgono la loro parte anche forze politiche ed economiche. Un certo costruttore vuole promulgare uno standard che si adatti alle sue apparecchiature: così alcuni fra gli standard attuali esistono perché sono stati offerti da un venditore che domina un mercato. D'altra parte, fattori tecnici come l'efficienza di determinati programmi che girano su determinate macchine perpetuano differenze che riguardano i particolari. Ci vorrà parecchio tempo prima che possano emergere standard universali e che gli utenti abbiano la possibilità di trasferire i loro testi da un sistema di elaborazione a un altro.

Lasciando da parte gli schemi di codificazione, c'è la forma delle lettere stesse. Su una tastiera di macchina per scrivere una A è semplicemente una A. Tipograficamente, invece, una A è una A o una *A* o una **A.** Nel nuovo campo della tipografia digitale il calcolatore è uno strumento per il disegno e la presentazione di forme di caratteri. Parte delle ricerche si rivolge alle forme in sé: in particolare alla rappresentazione di caratteri come strutture composite di punti e spazi. Altre ricerche si concentrano sulla formulazione di un codice per la memorizzazione al calcolatore di testi che combinino caratteri di fonti diverse (come il Times Roman e l'Helvetica) e di stili diversi (come il *corsivo* e il **neretto**).

Finora mi sono occupato solo di successioni memorizzate di caratteri, ma uno dei compiti principali di un programma di elaborazione di testi è quello di trattare margini e spaziature, cioè la «geografia» della pagina stampata. Nel linguaggio di composizione chiamato TEX vengono inseriti nel testo comandi che specificano caratteri non standard, modificano lo stile dei caratteri, fissano i margini e via dicendo *(si veda l'illustrazione in alto a pagina 80)*. Un comando di TEX si

Giovanni vuole sposare una norvegese.

$\exists x$ Norvegese(x) $\wedge$ Volere(Giovanni,[Sposare(Giovanni, x)])

Volere(Giovanni,[$\exists x$ Norvegese(x) $\wedge$ Sposare(Giovanni, x)])

Si ha ambiguità semantica quando un sintagma può svolgere ruoli diversi nel significato di una frase. Qui i ruoli del sintagma «una norvegese» sono resi espliciti quando la frase «Giovanni vuole sposare una norvegese» viene «trattata» in una forma logica basata sul calcolo dei predicati. Secondo un'interpretazione, chi pronuncia la frase ha in mente una persona ben determinata e ha scelto la sua nazionalità come mezzo per identificarla. Allora la frase significa: Esiste ($\exists$) una *x* tale che *x* è norvegese e ($\wedge$) *x* è la persona che Giovanni vuole sposare. Secondo un'altra interpretazione, né Giovanni né chi pronuncia la frase hanno in mente una persona specifica. Forse Giovanni ha in programma un viaggio in Norvegia e spera di incontrare in quel paese «un buon partito».

Lasciò cadere il piatto sul tavolo e lo ruppe.

Lasciò cadere il piatto sul tavolo e ruppe [il piatto].

Lasciò cadere il piatto sul tavolo e ruppe [il tavolo].

Si ha ambiguità pragmatica quando una frase può avere più di un significato per la presenza di una parola come il pronome «lo». Supponiamo che la frase riportata nell'illustrazione sia fornita a un calcolatore. Se la macchina può accedere a una conoscenza memorizzata nella grammatica delle frasi italiane, ma non ha accesso alle conoscenze del buon senso sulle proprietà di tavoli e piatti, il calcolatore potrebbe dedurre con pari validità che si sia rotto il tavolo o che si sia rotto il piatto.

distingue dal testo normale perché è racchiuso fra sbarre rovesciate (\). Il testo memorizzato viene «compilato» dal programma TEX, che interpreta i comandi inseriti al fine di creare un documento stampato secondo il formato specificato.

La compilazione è molto complessa, e spesso sono necessarie parecchie elaborazioni per passare dal codice creato mediante un programma di elaborazione testi al codice che in effetti pilota una stampante o una macchina da fotocomposizione. Un algoritmo che giustifichi il testo (cioè che provveda a chiudere perfettamente ogni riga, giocando con gli spazi a disposizione) deve determinare quante parole stanno in una riga, quanto spazio deve essere inserito fra le singole parole e se l'aspetto di una riga possa migliorare spezzando in sillabe una parola. L'algoritmo può anche provvedere a evitare difetti visivi, come una riga con ampi spazi fra una parola e l'altra seguita da una riga molto compatta. Il posizionamento di ciascuna riga sulla pagina è complicato ulteriormente dalla collocazione di intestazioni, note a piede di pagina, illustrazioni, tabelle e via dicendo. Le formule matematiche hanno poi le loro regole tipografiche speciali.

Il TEX e altri programmi simili sono primitivi per quel che riguarda un altro aspetto dell'elaborazione dei testi: l'interfaccia verso l'utente. Le unità video ad alta risoluzione che cominciano ora a essere disponibili permettono al calcolatore di visualizzare per l'utente, con buona approssimazione, la pagina che verrà stampata, con la posizione che ogni elemento andrà a occupare e i caratteri che verranno effettivamente utilizzati. Questo fa pensare che l'utente non dovrebbe essere costretto a inserire speciali successioni di comandi, ma possa invece manipolare la geografia della pagina direttamente sullo schermo per mezzo della tastiera e di un dispositivo di puntamento come un *mouse*. L'interfaccia fra calcolatore e utente che ne risulterebbe andrebbe a cadere in quella classe di interfacce denominate WYSIWYG, che sta per «*What you see is what you get*», ovvero «Quello che vedi è quello che ottieni».

Vale la pena di notare che i programmi per la manipolazione di testi prendono nomi diversi in ambiti professionali diversi. I programmatori parlano di *text editor*, ma fra gli utenti commerciali e nell'editoria vengono chiamati *word processor:* in quest'ultimo campo un *editor*, cioè un redattore, è chi ha il compito di migliorare la qualità di un testo. Comincia a vedersi anche software di ausilio per questo aspetto, più sostanziale, della redazione: software che non si riferisce né agli aspetti visivi del linguaggio né al contenuto concettuale, ma riguarda l'ortografia, la grammatica e lo stile. Fanno parte di questa classe due tipi diversi di programmi: opere di riferimento meccanizzate e strumenti meccanizzati per la verifica della correttezza.

Un esempio di opera di riferimento meccanizzata è un programma *thesaurus*, progettato in modo che, quando chi scrive indica una parola, sullo schermo appaia un elenco di sinonimi. Nei sistemi più perfezionati il *thesaurus* è totalmente integrato nel programma di elaborazione dei testi. Chi scrive colloca un contrassegno per evidenziare la parola da sostituire, quindi chiama in causa il *thesaurus* che visualizza le alternative in una «finestra» sullo schermo. Quando si sposta il contrassegno su una delle alternative, questa va automaticamente a sostituire la parola indesiderata.

Redigere un programma del genere comporta problemi tanto linguistici quanto computazionali. Sul versante linguistico, il meccanismo per la ricerca di una parola dovrebbe avere sufficiente flessibilità da accettare forme varianti. Per esempio, le informazioni memorizzate che si riferiscono al verbo «dotare» dovrebbero essere accessibili a richieste relative a «dotato», «dotando», «dota», e anche «dote» e «dotazione». Il riconoscimento di una radice comune in parole simili richiede un'analisi morfologica che può essere effettuata per mezzo di tecniche sviluppate nel corso delle ricerche sulla traduzione automatica. Fra i problemi computazionali rientra anche la formulazione di metodi per memorizzare ed effettuare una ricerca in un *thesaurus* o in un dizionario, che per poter essere davvero utile deve essere assai ampio.

Uno strumento di verifica della correttezza tratta l'ortografia, la grammatica e addirittura alcuni elementi di stile. I programmi più semplici cercano di mettere in corrispondenza ciascuna parola di un testo con un lemma in un dizionario memorizzato. Le parole che non trovano un corrispondente sono evidenziate come possibili errori di ortografia. Altri programmi ricercano errori grammaticali comuni o elementi stilisticamente infelici. Per esempio il Writer's Workbench, realizzato agli AT&T Bell Laboratories, comprende programmi che ricercano parole ripetute come «*the the*» (lo lo, o la la), un errore di battitura comune, punteggiature scorrette come «?.» ed espressioni verbose come «*at this point in time*» (in questo istante nel tempo). Un diverso strumento di verifica della correttezza richiama l'attenzione di chi scrive su espressioni «pompose» come «*exhibit a tendency*» (mostrare una tendenza) e «*arrive at a decision*» (arrivare a una decisione) e suggerisce sostituzioni con espressioni più semplici come «*tend*» (tendere) e «*decide*» (decidere). Un altro tipo di strumento, infine, va alla ricerca di termini di genere specifico come *mailman* (postino) e suggerisce la sostituzione con termini come *mail carrier* (portalettere).

Oltre a ricercare in un testo particolari successioni di caratteri, taluni programmi per la verifica della correttezza effettuano anche analisi statistiche. Calcolano la lunghezza media delle frasi, la lunghezza delle parole e grandezze simili, formulando infine un «indice di leggibilità». I passaggi con un indice di leggibilità scarso vengono portati all'attenzione di chi scrive. Non esistono ancora programmi in grado di effettuare un'ampia analisi grammaticale di un testo, ma un sistema sperimentale chiamato Epistle, realizzato

a

```
\inset
This is a sample of a {\italic justified} piece of text, which contains {\eightpoint small letters {\bold and }} {bigFont big ones}.
It includes foreign words such as \lquote pe\~na\rquote—which is Spanish—and foreign letters like \alpha\ and \aleph,
which can be baffling, and includes one \hskip 1.3in wide space.
```

b

01110100	01100101	01110010	01110011	00000000	00100111	00101101	11010011	00001000	01100001	01101110	01100100	00000000
t	e	r	s	NUOVA ENTITÀ	CODICE FONTE	POSI-ZIONE X	POSI-ZIONE Y	INCRE-MENTO X	a	n	d	NUOVA ENTITÀ

00110100	00110001	10110110	00101101	01100010	01101001	01100111	00100000	01101111	01101110	01100101	01101011	00101110
CODICE FONTE	POSI ZIONE X	POSI-ZIONE Y	INCRE-MENTO X	b	i	g	SPAZIO	o	n	e	s	•

00000000	00000001	10101111	10110110	00101100	01001001	01110100	00100000	01101001
NUOVA ENTITÀ	CODICE FONTE	POSI-ZIONE X	POSI-ZIONE Y	INCRE-MENTO X	I	t	SPAZIO	i

c

This is a sample of a *justified* piece of text, which contains small letters **and**
big ones. It includes foreign words such as "peña"—which is Spanish—and foreign letters like α and ℵ, which can be baffling, and includes one wide space.

L'elaborazione di testi *(word processing)*, cioè la preparazione e la correzione di testi con l'ausilio del calcolatore, richiede varie rappresentazioni del testo, perché il formato migliore per le interazioni fra il software e l'utente non è efficiente per l'invio di istruzioni a una stampante, né può fornire un'immagine preliminare in maniera efficiente del risultato della stampa. Nel linguaggio di composizione TEX il testo battuto dall'utente *(a)* comprende comandi che specificano caratteri non standard, modificano lo stile del carattere, fissano i margini e via dicendo. Questi comandi sono evidenziati da una sbarra rovesciata (\). Il programma TEX «compila» il testo fornito in ingresso, producendo un codice di calcolatore in grado di pilotare una macchina compositrice *(b)*. Per questo il codice viene diviso in «entità», ciascuna delle quali specifica il carattere e la posizione di partenza per una successione di parole. «Incrementi *X*» codificati spaziano le parole per riempire esattamente la distanza fra i margini sulla pagina stampata: «giustificano» così la riga tipografica. La pagina stampata *(c)* mostra il risultato. Il testo inglese dell'esempio significa «Questo è un campione di un pezzo di testo giustificato che contiene caratteri piccoli e grandi. Comprende parole straniere come *peña* - che è spagnolo - e lettere straniere come alfa e alef che lasciano perplessi; comprende inoltre un ampio spazio».

alla International Business Machines Corporation, può formulare qualche giudizio grammaticale. Usa una grammatica di 400 regole e un dizionario di 130 000 parole: anche questo sistema, come tutto il software che tenta di analizzare sintatticamente un testo senza affrontare ciò che il testo stesso significa, non può analizzare correttamente molte frasi.

Esiste software che affronti davvero il significato, cioè che presenti quel tipo di attività deduttiva che una persona userebbe nell'eseguire compiti linguistici come tradurre, riassumere o rispondere a una domanda? Software di questo tipo è l'obiettivo di progetti di ricerca nel campo dell'intelligenza artificiale fin dalla metà degli anni sessanta, quando cominciarono a essere disponibili le apparecchiature di elaborazione e le tecniche di programmazione necessarie, anche se risultava ormai evidente l'impossibilità pratica di una traduzione automatica. Vi sono, invece, molte applicazioni in cui software di questo tipo dimostrerebbe la sua utilità: programmi che accettino comandi in linguaggio naturale, programmi per il recupero delle informazioni, programmi che riassumano testi e programmi che acquisiscano conoscenze, basate sul linguaggio, per sistemi esperti.

Non esiste software che tratti con il significato per un sottoinsieme significativo dell'inglese (o di qualche altra lingua naturale): tutti i programmi sperimentali sono basati sull'identificazione di una versione semplificata della lingua e del significato e sulla verifica di quel che può essere fatto entro quei confini. Per alcuni ricercatori non esisterebbero barriere fondamentali alla stesura di programmi che presentino una piena comprensione del linguaggio naturale. Per altri, invece, la comprensione del linguaggio da parte del calcolatore è impossibile. Per capire le motivazioni degli uni e degli altri è importante prima esaminare i principi che starebbero alla base del funzionamento di un programma in grado di comprendere il linguaggio.

Un programma in grado di comprendere il linguaggio dovrebbe avere numerosi componenti, corrispondenti ai vari livelli di analisi del linguaggio *(si vedano le illustrazioni in queste due pagine e nelle pagine successive)*. La maggior parte dei programmi opera sul linguaggio scritto: pertanto l'analisi delle onde sonore è aggirata e il primo livello di analisi è morfologico. Il programma applica regole che scompongono una parola nella sua radice (cioè nella sua forma fondamentale) e in inflessioni come le terminazioni *-s* e *-ing* dell'inglese. Le regole corrispondono in gran parte alle regole di ortografia che i bambini imparano nelle scuole elementari. I bambini inglesi imparano, per esempio, che la radice di *baking* è *bake*, mentre quella di *barking* è *bark*. Un elenco delle eccezioni contiene le parole a cui le regole non si applicano, come le forme del verbo *be*. Altre regole associano le inflessioni con «caratteristiche» delle parole. Per esempio, *am going* è un verbo progressivo: segnala un'azione in divenire.

Per ciascuna radice che emerge dall'analisi morfologica un dizionario fornisce l'insieme delle categorie lessicali a cui la radice appartiene. Questo è il secondo livello di analisi eseguito dal calcolatore. Alcune radici (come *the*) hanno solo una categoria lessicale; altre ne hanno più d'una. *Dark* può essere un nome o un aggettivo; *bloom* può essere un nome o un verbo. In alcuni casi l'analisi morfologica limita le possibilità. (Nell'uso comune *bloom* può essere un nome o un verbo, mentre *blooming* è solo un verbo.) L'esito dell'analisi morfologica e lessicale è pertanto una successione di parole in una frase, dove ciascuna parola porta una certa quantità di informazioni di dizionario e di informazioni relative alle proprie caratteristiche. Questi dati costituiscono l'ingresso per il terzo componente del programma, l'analizzatore sintattico o *parser*, che applica le regole grammaticali per determinare la struttura della frase.

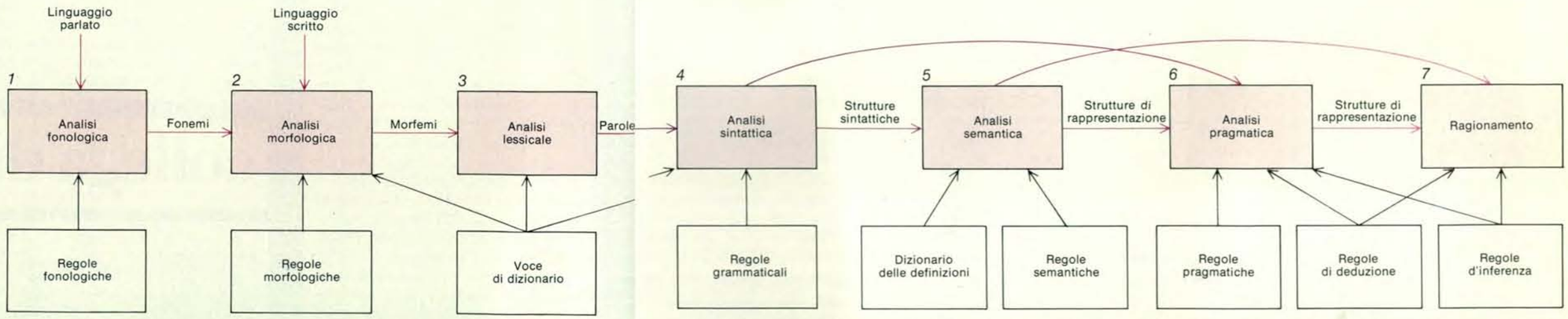
Nella costruzione di un buon analizzatore sintattico insorgono due problemi distinti. Il primo è quello di specificare un insieme preciso di regole, una grammatica, che determini l'insieme delle possibili strutture delle frasi di una lingua. Negli ultimi trent'anni molte ricerche di linguistica teorica si sono concentrate sulla formulazione di sistemi linguistici formali: costruzioni in cui le regole sintattiche di una lingua sono enunciate con tanta precisione da poter essere utilizzate da un calcolatore nella sua analisi. Le grammatiche generative trasformazionali di Noam Chomsky del Massachusetts Institute of Technology sono state il primo tentativo di ampio respiro: specificano la sintassi di una lingua mediante un insieme di regole la cui applicazione meccanica genera tutte le strutture accettabili.

Il secondo problema è quello dell'analisi sintattica stessa. Non è sempre possibile, quando si incontra una parte di frase, stabilire quale ruolo essa svolga nella frase completa o se le parole che la compongono vadano effettivamente associate. Prendiamo la frase *«Roses will be blooming in the dark gardens we abandoned long ago»* (Le rose fioriranno negli oscuri giardini che abbandonammo tanto tempo fa). Le parole *«in the dark»* potrebbero essere interpretate come un sintagma completo: tutto sommato, sono grammaticalmente ben formate e hanno un senso («nell'oscurità»). Ma il sintagma non può costituire un'unità coerente in un'analisi completa della frase, poiché costringe a interpretare *«Roses will be blooming in the dark»* come una proposizione («Le rose fioriranno nell'oscurità») lasciando *«gardens we abandoned long ago»* (i giardini che abbandonammo tanto tempo fa) senza alcun ruolo.

Gli analizzatori sintattici adottano varie strategie per esplorare i molteplici modi in cui i sintagmi possono essere combinati insieme. Alcuni procedono dall'alto verso il basso, tentando sin dall'inizio di identificare possibili frasi; altri procedono dal basso verso l'alto, saggiando combinazioni locali di parole. Alcuni retrocedono per esplorare in profondità le alternative, se una data possibilità fallisce; altri usano l'elaborazione in parallelo per seguire simultaneamente numerose alternative. Alcuni impiegano formalismi (come la grammatica trasformazionale) sviluppati dai linguisti; altri usano formalismi più recenti, progettati esplicitamente per i calcolatori. Questi ultimi formalismi sono più adatti per la realizzazione di procedure di analisi sintattica. Per esempio, le «reti di transizioni aumentate» esprimono la struttura di frasi e sintagmi come una successione esplicita di «transizioni» che debbono essere seguite da una macchina. Le «grammatiche a funzioni lessicali» creano invece una «struttura funzionale» in cui le funzioni grammaticali come testa, soggetto e oggetto sono collegate esplicitamente a parole e sintagmi che assolvono queste funzioni.

Non esiste una grammatica formale che possa trattare con successo tutti i problemi grammaticali di qualunque lingua naturale, ma le grammatiche e gli analizzatori sintattici esistenti possono risolvere più del 90 per cento di tutte le frasi. È un dato che va preso, però, con qualche cautela. Una data frase può avere centinaia, se non addirittura migliaia di analisi sintattiche possibili, la maggior parte delle quali non ammette un significato plausibile. Gli esseri umani non sono consapevoli del fatto che queste possibilità siano prese in considerazione e rifiutate, mentre i programmi di analisi sintattica restano impantanati in alternative senza senso.

L'uscita di un programma di analisi sintattica diventa l'ingresso per il quarto componente di un programma in grado di



Per poter dire che comprende il linguaggio, un calcolatore deve potersi basare su vari tipi di dati memorizzati *(riquadri bianchi)* e deve poter eseguire numerosi livelli di analisi *(riquadri in colore)*. Nel caso del linguaggio parlato, la prima analisi è quella fonologica *(1)*: il calcolatore analizza le onde sonore. Se il linguaggio è scritto, la prima analisi è quella morfologica *(2)*: il calcolatore scompone ogni parola nella sua radice, cioè nella sua forma fondamentale, e nelle inflessioni. Segue l'analisi lessicale *(3)*, in cui il calcolatore assegna le singole parole alla relativa categoria lessicale (nome, aggettivo ecc.) e ne indentifica «caratteristiche» come il genere o il numero. Viene quindi l'analisi sintattica *(4)*: l'applicazione di regole grammaticali per ricostruire la struttura della frase. Nell'analisi semantica *(5)*, la frase viene trasformata in una forma che consenta di trarre inferenze. Lo stadio finale è l'analisi pragmatica *(6)*, che esplicita il contesto della frase, cioè elementi come il rapporto fra il momento in cui viene emessa e il momento a cui si riferisce. A questo punto il calcolatore è in grado di trarre inferenze *(7)*, eventualmente per formulare una risposta alla frase.

comprendere il linguaggio: un analizzatore semantico, che traduce la forma sintattica di una frase in una forma «logica». L'obiettivo è quello di porre le espressioni linguistiche in una forma che consenta al calcolatore di applicare procedure di ragionamento e di trarne inferenze. Qual è la rappresentazione più adeguata? Anche qui esistono teorie rivali. Come già per l'analisi sintattica, i problemi chiave riguardano l'efficacia e l'efficienza.

L'efficacia dipende dall'identificazione delle strutture formali adeguate per codificare il significato di espressioni linguistiche. Una possibilità è offerta dal calcolo dei predicati, che usa i quantificatori ∀ per indicare «tutti» ed ∃ per indicare «esiste». Nel calcolo dei predicati, *«Roses will be blooming...»* è equivalente a «Esiste un qualcosa che è una rosa e che fiorisce...». Ma c'è una difficoltà. Una rosa è adeguata per rappresentare il significato di *«Roses will be blooming»*, o sarebbe meglio specificare che si tratta di due o più rose? Come può decidere il calcolatore? Il dilemma diventa più grave se una frase comprende un nome non numerabile come «acqua», per esempio: «l'acqua scorrerà...». Non si può parlare di un'acqua, due acque... come si parla di una casa, due case... Nel progettare una struttura formale per il significato di espressioni linguistiche, l'implicita vaghezza del linguaggio naturale genera molti problemi analoghi.

Bisogna considerare anche l'efficienza, perché il calcolatore userà la forma logica di una frase per trarre inferenze che, a loro volta, serviranno sia per l'analisi del significato della frase, sia per formulare una risposta a essa. Alcuni formalismi, come il calcolo dei predicati, non possono essere ricondotti direttamente a un'elaborazione efficiente, ma sono state formulate altre rappresentazioni più «procedurali». Supponiamo che si debba rispondere alla domanda: «Ci sono fiori nei giardini che abbiamo abbandonato tanto tempo fa?». Il calcolatore deve sapere che le rose sono fiori, conoscenza che potrebbe essere rappresentata da una formula del calcolo dei predicati equivalente all'affermazione «Tutto ciò che è una rosa è anche un fiore». Poi, per trarne la dovuta deduzione, il calcolatore potrebbe applicare tecniche sviluppate per la dimostrazione automatica di teoremi. Un modo diverso di affrontare il problema sarebbe quello di attribuire uno statuto computazionale privilegiato a determinate inferenze. Per esempio, le deduzioni fondamentali relative a classificazioni potrebbero essere rappresentate direttamente in strutture di dati *(si veda l'illustrazione in basso a pagina 84)*. Deduzioni di tal genere sono costantemente necessarie per ragionare sulle proprietà ordinarie degli oggetti. Altri tipi di fatti (per esempio, che i fiori hanno bisogno di acqua per crescere) potrebbero poi essere rappresentati in una forma più vicina al calcolo dei predicati. Il calcolatore potrebbe basarsi su ambedue gli strumenti per trarre inferenze (per esempio, che se le rose non ricevono acqua non crescono).

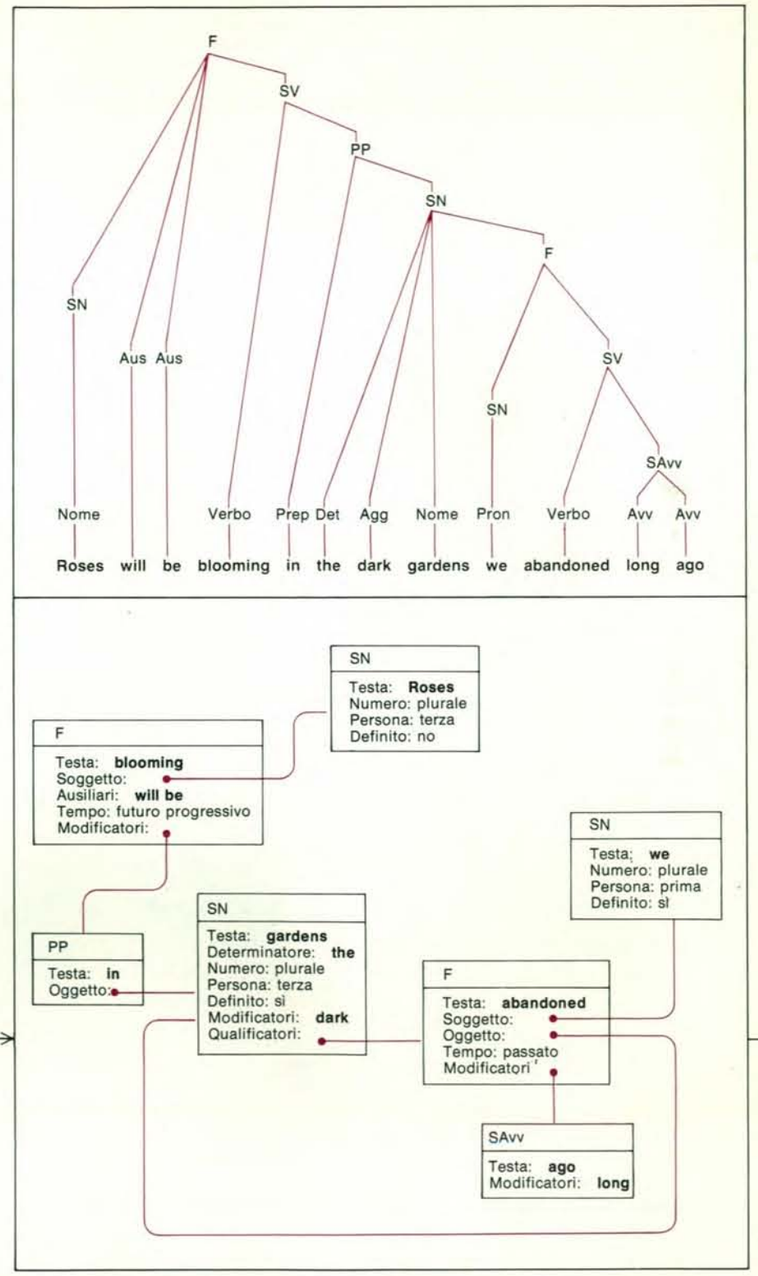
Molte ricerche si sono rivolte alla progettazione di «linguaggi di rappresentazione» in grado di fornire una codificazione efficace ed efficiente del significato. La difficoltà maggiore sta nella natura del ragionamento umano basato sul buon senso. La maggior parte delle conoscenze di una persona non può essere formulata in regole logiche del tipo «tutto o nulla»: in genere si tratta di «attese normali». Se si chiede: «C'è sporco in giardino?», la risposta è quasi sicuramente «sì». Questo sì, tuttavia, non può essere un'inferenza logica: alcuni giardini sono idroponici e in essi le piante crescono nell'acqua. Una persona tende a fare affidamento sulle attese normali senza pensare alle eccezioni, fuorché quando sono rilevanti, ma purtroppo non sono stati fatti molti progressi verso una formalizzazione del concetto di «rilevanza» e del modo in cui questo dà forma a quel sottofondo di attese che incide regolarmente sulla comprensione delle espressioni linguistiche.

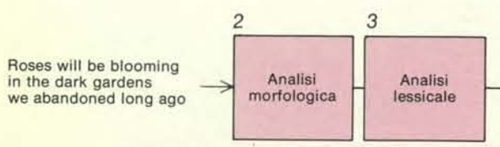
Lo stadio finale dell'analisi in un programma in grado di comprendere il linguaggio è l'analisi pragmatica, cioè l'analisi del contesto. Le frasi non galleggiano nel vuoto: sono pronunciate da una persona particolare in un particolare istante e fanno riferimento, almeno implicitamente, a un particolare insieme di conoscenze. L'inserimento nel contesto è, in parte, diretto: il pronome «io» si riferisce al parlante; l'avverbio «ora» si riferisce al momento in cui la frase viene pronunciata. Ma anche queste parole possono essere problematiche: pensate all'uso di «ora» in una lettera che io scrivo oggi pensando che il destinatario la leggerà fra tre o quattro giorni. Anche qui, però, programmi non molto complicati possono trarre, nella maggior parte dei casi, la conclusione giusta. Altri aspetti dell'inserimento nel contesto sono più complessi. Il pronome «noi» è un esempio. «Noi» può riferirsi al parlante e all'ascoltatore o al parlante e a qualche terza persona. Quale sia l'alternativa giusta (ed eventualmente quale sia la terza persona) non è detto esplicitamente e, in effetti, costituisce una tipica fonte di incomprensioni nella normale conversazione fra esseri umani.

Altri tipi di inserimenti nel contesto non sono segnalati da una parola problematica come «noi». Una frase come «Le rose fioriranno...» presuppone l'identificazione di qualche momento futuro in cui le rose saranno effettivamente in fiore. Così, quella frase potrebbe essere stata il seguito di una frase del tipo «Come sarà quando arriveremo a casa?» oppure «L'estate sta per arrivare». Analogamente, il sintagma nominale «gli oscuri giardini che abbandonammo molto tempo fa» ha un significato dipendente dal contesto. Può darsi che esista solo un caso di giardini in cui siamo stati insieme; ma magari, no. La frase presuppone un insieme di conoscenze sulla base del quale i giardini sono identificabili. Il fatto è che un sintagma che inizi con l'articolo determinativo raramente determina a pieno l'oggetto a cui si riferisce.

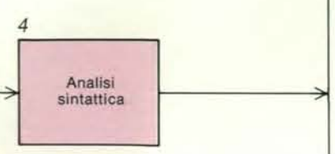
Come trattare questi sintagmi? Si potrebbe codificare la conoscenza del mondo in una forma che il programma può utilizzare per trarne inferenze. Per esempio, dalla frase «Sono andato al ristorante e il cameriere era scortese» si può dedurre che «il cameriere» si riferisce alla persona che ha servito il pasto a chi parla, se la nostra conoscenza comprende, per così dire, una scaletta degli eventi che tipicamente costituiscono un pranzo in un ristorante. (Ciascun avventore è servito da un particolare cameriere o da una particolare cameriera.) In casi più complessi può essere d'aiuto un'analisi degli obiettivi e delle strategie del parlante. Se si sente dire «Ho l'esame di matematica domani, dov'è il mio libro?» si può ipotizzare che il parlante abbia intenzione di studiare e che «il mio libro» significhi il testo di matematica usato nel corso che il parlante ha seguito. Questa impostazione va incontro alla stessa difficoltà che ostacola la rappresentazione del significato, la difficoltà di formalizzare quel buon senso di fondo che determina quali scalette, quali obiettivi e quali strategie siano rilevanti e in che modo interagiscano. I programmi che sono stati scritti finora funzionano solo in campi estremamente artificiali e limitati e non è affatto chiaro fino a che punto possano essere estesi.

Ancora più problematici sono gli effetti del contesto sul significato delle parole. Supponiamo che nell'incontrare l'espressione «gli oscuri giardini che abbiamo

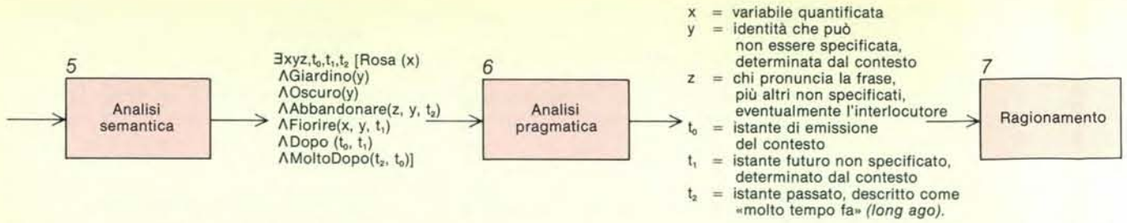




Parola	Radice	Categorie lessicali	Caratteristiche
Roses	rose	Nome	[plurale]
will		Verbo (ausiliare)	[modale]
be		Verbo (ausiliare)	[infinito]
		Verbo (copulativo)	[infinito]
blooming	bloom	Verbo (intransitivo)	[progressivo]
in		Preposizione	
the		Determinatore	[definito]
dark		Aggettivo	
		Nome	[non numerabile]
gardens	garden	Nome	[plurale]
		Verbo	[terza persona, singolare, presente]
we		Pronome	[prima persona, plurale, nominativo]
abandoned	abandon	Verbo (transitivo)	[passato]
		Verbo (transitivo)	[participio]
long		Aggettivo	
ago		Avverbio	



Questa successione di analisi eseguite da un ipotetico programma di calcolatore ci dà un'idea di come funzioni il software in grado di comprendere il linguaggio. In questa illustrazione è stata fornita in ingresso al programma la frase inglese *«Roses will be blooming in the dark gardens we abandoned long ago»* («Le rose fioriranno negli oscuri giardini che abbandonammo tanto tempo fa»). Le prime analisi (morfologica e lessicale) danno un elenco delle parole che compongono la frase, con le relative radici, categorie sintattiche e caratteristiche. I dati fungono da ingresso per il livello sintattico dell'analisi. Qui la struttura superficiale viene posta in forma di albero. Presumibilmente il calcolatore prende in considerazione e scarta numerosi alberi sbagliati. Per esempio, scarta un albero in cui *«Roses will be blooming in the dark»* risulta una frase completa. La struttura profonda della frase è presentata quindi come un diagramma di struttura funzionale, in cui le relazioni fra le parti della frase sono rese esplicite e visualizzate come linee di congiungimento fra i riquadri. Alcune relazioni erano esplicite nella struttura superficiale (per esempio che *roses* è il soggetto di *blooming*), altre no (per esempio che *gardens* è il complemento oggetto di *abandoned*). Il risultato dell'analisi sintattica viene trasferito agli stadi finali del programma, presentati nell'illustrazione in alto della pagina successiva.



L'analisi si conclude con la trasformazione della struttura sintattica della frase *«Roses will be blooming...»* in una forma sulla base della quale il calcolatore può trarre inferenze. In questo esempio la trasformazione è basata sul calcolo dei predicati: così il modulo di analisi semantica del nostro software ipotetico rappresenta i contenuti logici della frase per mezzo di simboli che possono essere tradotti in linguaggio ordinario come «*x* è una rosa e *y* è un giardino e *y* è oscuro...». [Dato che il calcolo dei predicati dà strutture logiche, abbiamo reso gli elementi in italiano.] Infine, il modulo di analisi pragmatica specifica le informazioni note sulle variabili *x, y, z,* t_0, t_1 e t_2. La variabile *x*, per esempio, è «quantificata»: dichiara l'esistenza di qualcosa, anziché identificare un oggetto particolare. In altre parole il calcolatore interpreta *roses* come un termine che si riferisce alle rose in generale e non a rose particolari. Quindi *roses* non è un nome «definito». (Questa decisione è stata presa nel corso dell'analisi semantica.) Invece, la variabile rimane ambigua perché sta per il pronome ambiguo *we* (noi).

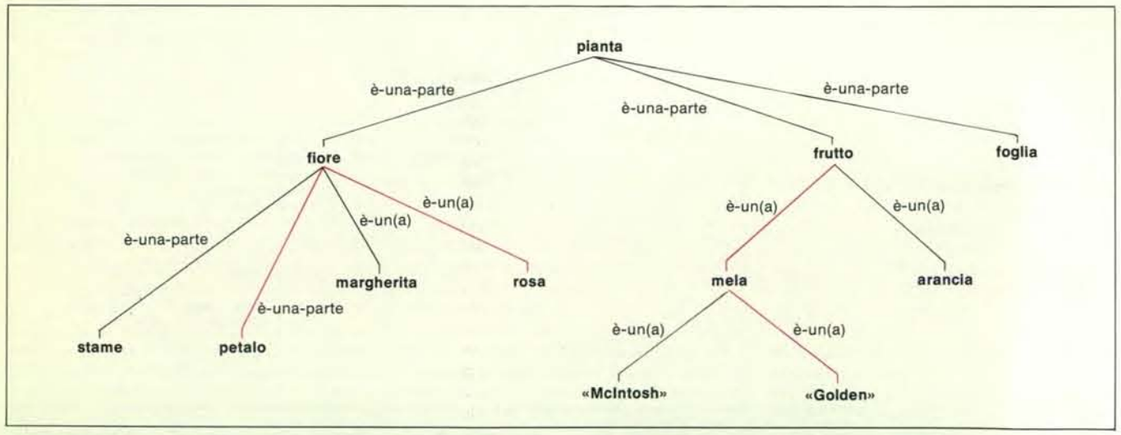
abbandonato tanto tempo fa» si cerchi di attribuire un particolare significato a «oscuri». E quale? Quello che ha in «i giorni oscuri del tormento» o quello di «Gli oscuri personaggi che si muovevano dietro a quella faccenda?» o ancora quello di «I suoi motivi ci sono oscuri»? C'è sempre un nucleo di analogia in tutti gli usi di una parola, ma il suo significato pieno è determinato dal modo in cui è usata e dalla conoscenza precedente che il parlante presume che esista nell'ascoltatore. Può darsi che «gli oscuri giardini» abbia un significato molto specifico per l'interlocutore, ma per tutti noi rimane un'espressione un po' misteriosa.

A tutta prima sembra possibile distinguere gli usi «letterali» del linguaggio da quelli metaforici o poetici. Allora, i programmi di calcolatore che dovessero confrontarsi esclusivamente con il linguaggio letterale non sarebbero intralciati dai dilemmi del contesto. Il problema è che la metafora e il «significato poetico» non sono confinati nelle pagine letterarie. Il linguaggio di tutti i giorni è pervaso da metafore inconsce, come quando si dice «Ho perso due ore per far passare la mia idea». Praticamente tutte le parole hanno un campo aperto di significati che sfumano gradualmente da quelli che sembrano perfettamente letterali a quelli che sono chiaramente metaforici.

Per ora (e con tutta probabilità per sempre) i limiti della formalizzazione del significato contestuale impediscono di progettare programmi di calcolatore che si avvicinino a una completa imitazione della comprensione umana del linguaggio. Gli unici programmi usati oggi nella pratica, e che tentano una sia pur limitata comprensione, sono «terminali» in linguaggio naturale che consentono all'utente di un programma di richiedere informazioni formulando domande in inglese (o in qualche altra lingua). Il programma risponde con frasi inglesi (o in quella data lingua) o con la visualizzazione di dati.

Uno dei primi esempi è stato un programma chiamato SHRDLU, sviluppato verso la fine degli anni sessanta. Esso consente a una persona di comunicare in inglese con un calcolatore a proposito di un mondo simulato di blocchi su un tavolo. Il programma analizza le richieste, i comandi e gli enunciati dell'utente e risponde con parole adeguate o con opportune azioni eseguite sulla scena simulata. SHRDLU funziona bene, in parte perché l'universo della sua conversazione è limitato a un'area semplice e specializzata: i blocchi e le poche azioni che si possono esercitare su di essi.

Più di recente sono state progettate alcune interfacce terminali pensando ad applicazioni pratiche. Una persona che vuole accedere a informazioni memorizzate nel calcolatore batte alla tastiera frasi in linguaggio naturale che il calcolatore interpreta come domande. La gamma delle domande è circoscritta dalla gamma dei



Una rete semantica è una forma particolare di struttura di archiviazione dei dati, che rappresenta le relazioni logiche in modo che un calcolatore possa trarne efficacemente taluni tipi di inferenza. Qui, semplicemente seguendo i collegamenti instaurati nella rete *(in colore)*, si ottiene l'inferenza che una «Golden» è un frutto e che una rosa ha petali. I fatti che non possono essere rappresentati facilmente e utilmente mediante una rete semantica di questo tipo possono essere rappresentati in altri modi, per esempio mediante il calcolo dei predicati.

dati sulla cui base sono formulate le risposte: in questo modo si può attribuire un significato preciso a ogni parola. In una base di dati relativa ad automobili, per esempio, «scuro» può essere definito come colore «nero» o «blu» e niente altro. Il significato contestuale c'è, ma è predeterminato da chi ha costruito il sistema, e si presume che l'utente lo debba apprendere.

Il vantaggio principale di un terminale in linguaggio naturale è quello di presentare ai potenziali utenti una barriera iniziale bassa. Se si invita una persona a porre una domanda nella lingua che parla tutti i giorni, quella persona sarà ben disposta a tentare, e se il calcolatore si dimostra incapace di manipolare la specifica forma della domanda, l'utente con tutta probabilità sarà disposto a modificarla fino a che non funziona e, con il tempo, imparerà a conoscere i vincoli imposti dal sistema. Al contrario, una persona che debba apprendere un linguaggio specializzato per poter porre una domanda avrà con tutta probabilità l'impressione di essere costretto a una quantità di lavoro sproporzionata.

Per finire, voglio parlare di un tipo di sistema relativamente nuovo, che viene chiamato «coordinatore». In sostanza, sostituisce la posta elettronica comune con un processo che coadiuva la generazione di messaggi e controlla il progredire delle conversazioni che ne risultano. I coordinatori sono basati sulla teoria degli atti linguistici, secondo la quale ogni emissione linguistica cade in una categoria che fa parte di un piccolo numero di categorie. Alcuni atti linguistici sono enunciati: «Sta piovendo». Alcuni sono atti espressivi: «Mi dispiace di averti pestato un piede». Alcuni sono richieste: «Per favore, portale il sacco» o «Come ti chiami?». Alcuni sono impegni: «Lo farò domani». Alcuni sono dichiarativi: «Sei licenziato». (Gli atti dichiarativi si distinguono dagli enunciati perché hanno un effetto per il solo fatto di essere stati effettuati.)

La classificazione degli atti linguistici è utile perché gli atti delle varie categorie non si svolgono a caso. Ogni atto linguistico ha «condizioni di felicità» sotto le quali risulta appropriato, e «condizioni di soddisfacimento», sotto le quali è completato. Per esempio, una richiesta o un impegno portano con sé, implicitamente o esplicitamente, l'idea di un tempo entro il quale la richiesta o l'impegno debbono essere soddisfatti. Inoltre, ogni atto linguistico fa parte di una conversazione che segue un andamento regolare. La regolarità è cruciale per una comunicazione che abbia successo.

Come per ogni aspetto del linguaggio, la piena comprensione di un dato atto linguistico è sempre intrecciata a quello sfondo inarticolato di attese del parlante e dell'ascoltatore. L'atto linguistico «Sarò qui domani» può essere una previsione o una promessa, e «Giochi a tennis?» può essere una domanda o un invito. Nella conversazione a viva voce l'intonazione e l'enfasi svolgono un ruolo importante nello stabilire questo significato.

I sistemi coordinatori trattano gli atti linguistici incorporati nei messaggi, specificando che cosa deve essere fatto e quando. Il sistema non tenta di analizzare il contenuto linguistico dei messaggi. Invece il software per l'elaborazione di testi al terminale del mittente chiede al mittente stesso di rendere esplicito il contenuto in atti linguistici di ciascun messaggio. Una persona può scrivere «Sarà un piacere per me farti avere quella relazione» nel messaggio stesso, ma deve aggiungere (premendo alcuni tasti speciali) che il messaggio è l'accettazione (ACCEPT) di una particolare richiesta (REQUEST). Il sistema quindi può conservare traccia dei messaggi e delle loro interconnessioni. In particolare, può controllare il completamento delle conversazioni, richiamando l'attenzione degli utenti in casi in cui sia in sospeso qualcosa di immediato o in cui non sia stato rispettato un tempo concordato per il soddisfacimento di qualche impegno.

In una prospettiva ampia, i coordinatori sono solo un elemento di una grande famiglia di software che fornisce agli utenti un mezzo strutturato in cui il linguaggio è ampliato con indicazioni esplicite dei collegamenti fra i vari elementi. Un altro tipo di software della stessa famiglia fornisce strumenti per riassumere e indicizzare documenti. Un altro tipo di software ancora è un albo elettronico che consente agli utenti di memorizzare e ricevere messaggi non indirizzati a un destinatario specifico. I messaggi sono «impostati» con una struttura aggiuntiva che ne indica il contenuto e ne facilita il reperimento da parte di lettori interessati.

La previsione più scontata, per il futuro del software che manipola il linguaggio, è che la diminuzione dei costi dell'hardware renderà largamente disponibili applicazioni oggi possibili ma non realizzabili in pratica. Tuttavia, software che imiti la piena comprensione umana del linguaggio non è affatto qualcosa di imminente. Si possono segnalare alcune tendenze specifiche.

La prima è che si tenderà a porre maggiormente l'accento sul linguaggio parlato. A dire il vero, la comprensione automatizzata del linguaggio parlato presenta tutte le difficoltà della comprensione del linguaggio scritto, e qualcuna in sovrappiù. La semplice operazione di articolare un'emissione nelle parole che la compongono può mettere a dura prova un calcolatore: le speranze per una «macchina per scrivere vocale» che scriva un testo sotto dettatura sono pallide come quelle per la traduzione automatica di alta qualità e per la comprensione del linguaggio. Molti dispositivi utili, però, non richiedono l'analisi del parlato continuo. Esistono sistemi che possono identificare una parola o una espressione formulate a voce sulla base di un vocabolario prefissato di poche centinaia di elementi: questi sistemi miglioreranno l'interfaccia fra utenti e macchine. La comparsa recente di chip di circuiti integrati a basso costo che elaborano segnali acustici favorirà sicuramente questa tendenza. I sintetizzatori di voce che generano emissioni comprensibili (anche se non molto naturali) avranno un ruolo sempre più importante. Tecniche perfezionate di «compressione» e di codificazione del parlato diffonderanno i messaggi acustici e l'annotazione acustica degli archivi automatizzati.

Una seconda tendenza, nel software che tratta il linguaggio, sarà un'attenzione sempre crescente e un'analisi teorica sempre più perfezionata dei vincoli che gravano in campo linguistico. Più volte, in questo articolo, ho fatto notare casi in cui i calcolatori trattano il significato in modo accettabile perché operano in un ambito limitato di significati possibili. Chi usa software di questo genere si rende conto ben presto che il calcolatore non capisce tutto il linguaggio, ma il sottoinsieme disponibile resta una buona base per la comunicazione. In gran parte, il successo commerciale del futuro software che tratti il linguaggio dipenderà dalla scoperta di campi in cui i vincoli su ciò che le frasi possono significare lasceranno ancora a disposizione dell'utente un'ampia gamma linguistica.

Una terza tendenza sta nello sviluppo di sistemi che uniscono il naturale e il formale. Spesso si dà per scontato che il modo migliore di comunicare con i calcolatori sia il linguaggio naturale. I progetti per una «quinta generazione» di calcolatori intelligenti si basano su questa convinzione. Non è affatto evidente che si tratti di una convinzione valida. In alcuni casi anche la più piena comprensione del linguaggio naturale non ha lo stesso potere espressivo di un'immagine e in molti casi una comprensione parziale del linguaggio naturale si dimostra meno utilizzabile di un'interfaccia formale ben progettata. Consideriamo il lavoro con terminali in linguaggio naturale. Qui il linguaggio naturale favorisce inizialmente l'accettazione del sistema, ma poi gli utenti spesso si spostano verso forme stilizzate di linguaggio che scoprono di poter usare tranquillamente, cioè senza doversi preoccupare se la macchina interpreterà i loro enunciati correttamente o no.

I sistemi che oggi hanno maggior successo facilitano questo passaggio. Alcuni (i coordinatori, fra gli altri) fondono il linguaggio naturale e quello formale: l'utente impara a riconoscere le proprietà formali delle emissioni e a includerle esplicitamente nei messaggi. Così il calcolatore manipola strutture formali, mentre le persone trattano compiti in cui il contesto è importante e non si possono applicare regole precise. Altri sistemi incorporano un sistema di interrogazione fortemente strutturato, cosicché a mano a mano che acquista esperienza l'utente vede che le forme artificiali risparmiano tempo e fastidi. In ogni caso non si assegnano al calcolatore i compiti difficili e aperti dell'analisi linguistica: la macchina funge invece da mezzo linguistico strutturato. Questo è forse il modo più utile in cui il calcolatore tratterà il linguaggio naturale.

Software per la grafica

Non più esclusivo dominio degli specialisti, la grafica interattiva al calcolatore sta rapidamente diventando il mezzo convenzionale di comunicazione tra il calcolatore e la maggior parte degli utenti

di Andries van Dam

Ivan E. Sutherland, un pioniere nella programmazione di calcolatori per la creazione e la manipolazione di immagini, sottolineò una volta, a proposito del suo argomento preferito: «Penso allo schermo di un calcolatore come a una finestra sul "paese delle meraviglie" nella quale un programmatore può descrivere oggetti che obbediscono alle ben note leggi di natura, ma anche oggetti puramente immaginari che seguono le leggi che egli ha inserito nel suo programma. Grazie a questi schermi sono atterrato su una portaerei in movimento, ho osservato una particella nucleare urtare una buca di potenziale, ho volato su un razzo a una velocità prossima a quella della luce e ho guardato un calcolatore svelare le sue attività più intime».

Fino a poco tempo fa l'esperienza di Sutherland relativa ai poteri apparentemente magici della grafica interattiva al calcolatore (chiamata anche «eidomatica», da *eidos*, immagine, e informatica) poteva essere condivisa soltanto da un piccolo gruppo di operatori, soprattutto scienziati e ingegneri impegnati nella progettazione assistita dal calcolatore (CAD), nell'analisi dei dati e nella costruzione di modelli matematici. Adesso il privilegio di esplorare mondi reali e immaginari attraverso lo specchio del calcolatore sta diventando sempre più comune. In effetti la grafica si sta avviando a essere la forma convenzionale di comunicazione con i calcolatori.

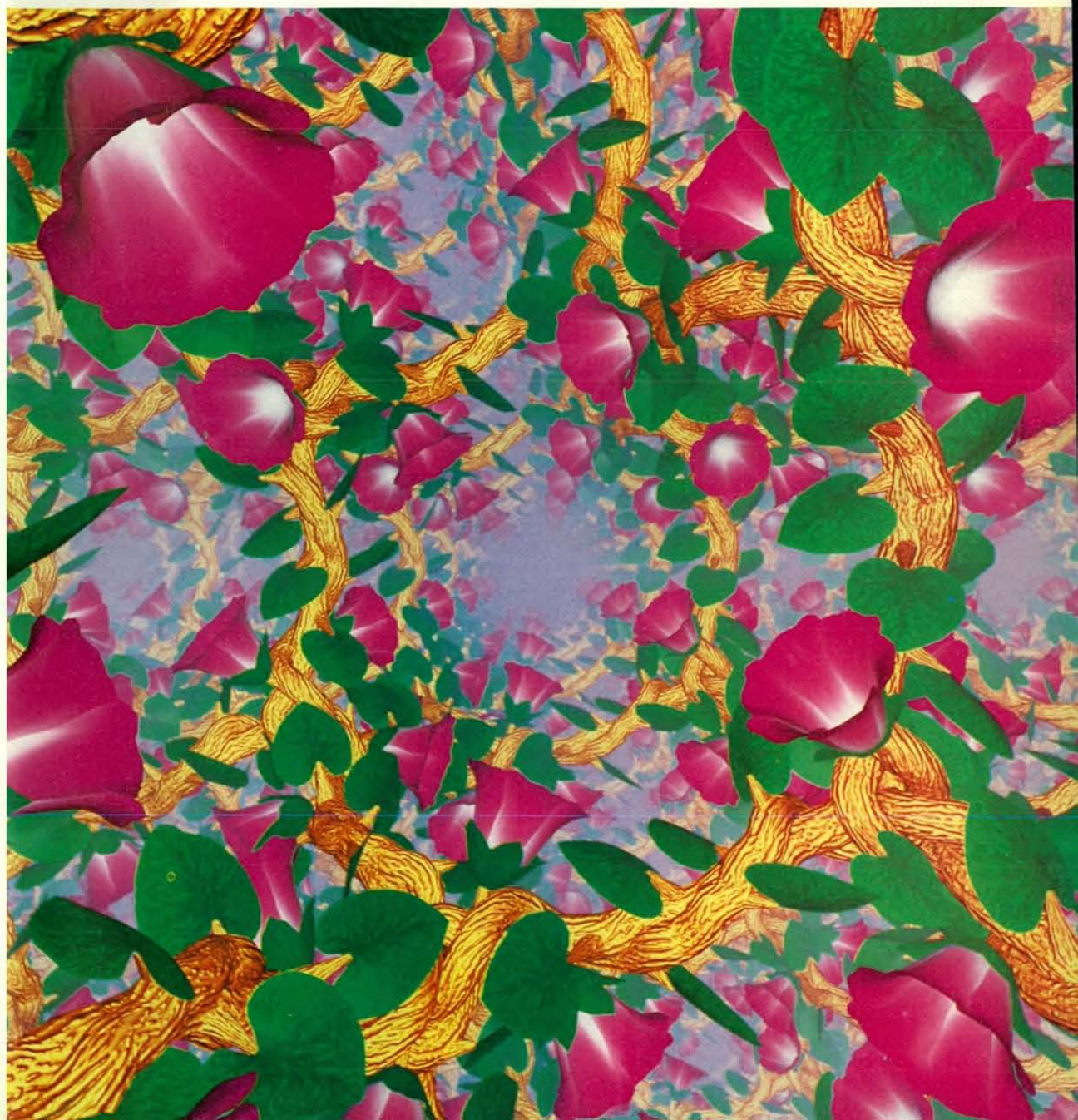
Questa nuova tendenza è dovuta a diversi motivi. Anzitutto i notevoli miglioramenti nel rapporto costo/prestazioni di taluni componenti dell'hardware del calcolatore hanno reso possibile un'ampia disponibilità di sofisticati terminali grafici e di personal computer con capacità grafiche. In particolare, passi avanti nella progettazione e nella fabbricazione di circuiti microelettronici hanno portato a una nuova generazione di «*chip*» di memoria, che offrono una grandissima capacità di immagazzinamento di dati a un costo unitario molto basso. Questo sviluppo ha a sua volta reso economicamente competitiva la grafica a *raster*, che si basa su una scansione sistematica a righe orizzontali analoga a quella dello schermo televisivo. Nei sistemi di grafica a scansione orizzontale ogni pixel (elemento di immagine) è rappresentato singolarmente nella memoria del calcolatore e quindi può essere controllato dal software in modo indipendente, fornendo al programmatore la massima flessibilità nella creazione e nella manipolazione di immagini.

Nel frattempo analoghi miglioramenti nel software hanno di gran lunga esteso la gamma di applicazioni che possono essere svolte con l'ausilio della grafica. Nuovi pacchetti di software per applicazioni commerciali consentono per esempio di visualizzare dati in forma di diagrammi e di grafici persino su piccoli ed economici calcolatori per uso domestico. Pacchetti di software di alto livello per uso grafico stanno inoltre diventando sempre più disponibili, facilitando così la scrittura e il trasferimento su più calcolatori di nuovi programmi applicativi.

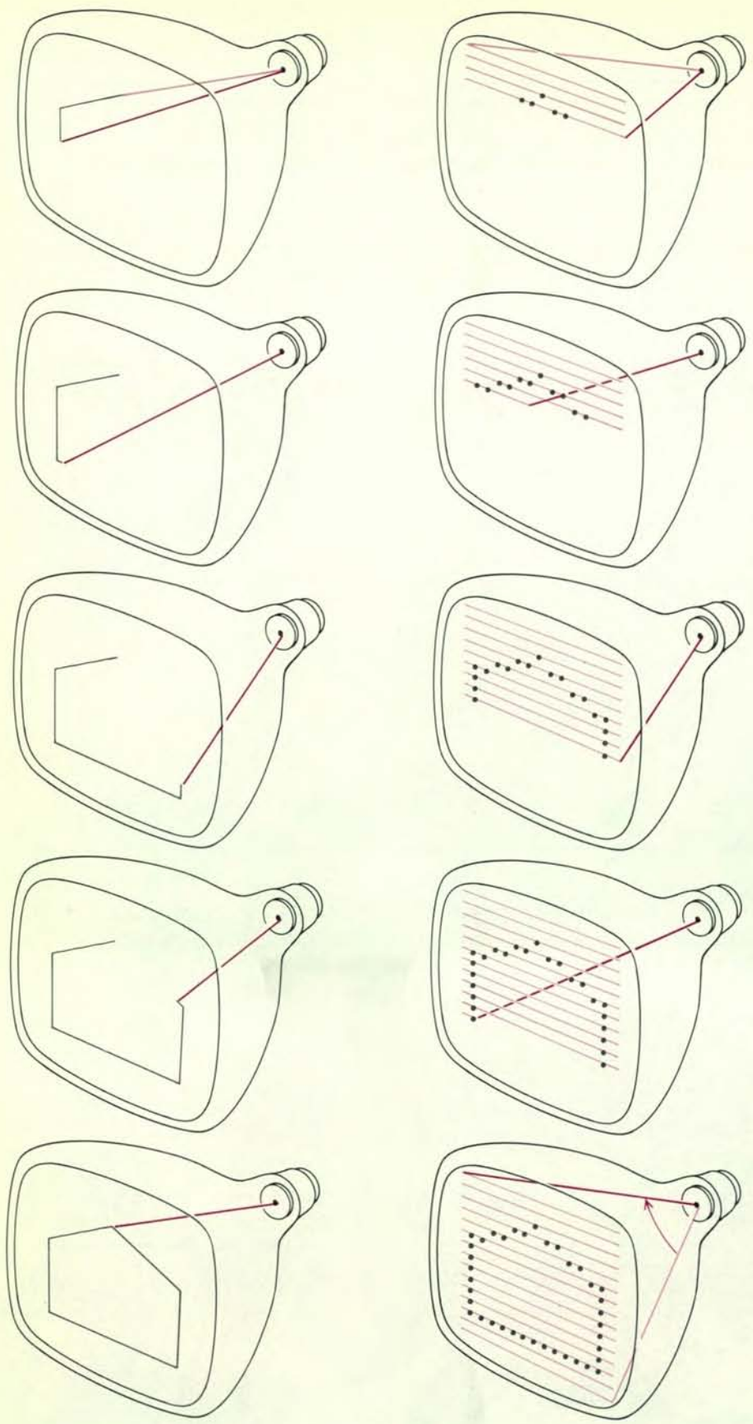
Un altro fattore determinante dell'aumento di popolarità della grafica al calcolatore è il modo in cui i sistemi di visualizzazione del calcolatore contribuiscono a quella che viene definita interfaccia operatore-macchina «amichevole verso l'utente». Questa espressione così elaborata si riferisce a una filosofia di progettazione del software meglio esemplificata da una serie di tecniche sviluppate negli anni settanta presso il Palo Alto Research Center della Xerox Corporation. Schermi di calcolatore con questa impostazione (influenzata dal precedente lavoro svolto dal gruppo di Douglas C. Engelbart allo Stanford Research Institute) sono ora disponibili in prodotti commerciali che vanno dalla stazione di lavoro Star della Xerox al personal computer Macintosh della Apple. Una caratteristica rilevante di questo tipo di interfaccia utente è quella di sfruttare la metafora della scrivania: il visualizzatore è diviso in regioni separate e possibilmente sovrapponibili chiamate «finestre», che si possono pensare come fogli di carta sparsi su un tavolo da lavoro. Ogni finestra può servire per la presentazione di un programma applicativo diverso, cosicché è possibile lavorare simultaneamente con testi e immagini e con l'aiuto di operazioni simulate di *collage* si possono riunire questi diversi elementi in un documento singolo.

I nuovi sistemi «amichevoli verso l'utente» si basano generalmente sul WYSIWYG («What you see in what you get», quello che vedi è quello che ottieni), un metodo con il quale lo schermo visualizza il più fedelmente possibile ciò che alla fine verrà stampato (o trasferito su altro supporto permanente). Per impaginare un testo, per esempio, non si devono immettere codici speciali di controllo (come ad esempio *.pp* per «paragrafo», o *.s2* per «salta due righe») che poi devono essere interpretati da un programma separato di impaginazione una volta che l'utente abbia finito il lavoro di redazione. Margini e rientri sono invece regolati utilizzando il facsimile di un righello con i segni dei vari punti di arresto e il testo viene continuamente reimpaginato in funzione delle indicazioni volta a volta date. Poiché i caratteri sono interamente generati da software grafico possono essere visualizzati in quasi ogni corpo o fonte, nonché distanziati in modo uguale o proporzionale alla loro larghezza. Allo stesso modo possono venire trattati simboli matematici, alfabeti differenti da quello latino e persino ideogrammi cinesi o giapponesi.

I programmi applicativi di tali sistemi si basano su un insieme uniforme di convenzioni per la specificazione dei comandi. Per esempio, anziché battere una serie di comandi su una tastiera alfanumerica, si può scegliere fra vari «menù», o liste di comandi, che appaiono sullo schermo. Un comando viene eseguito semplicemente indicandolo con un dispositivo, quale una penna ottica o un «*mouse*» (un meccanismo che si fa scivolare sul piano di lavoro per spostare un indicatore sullo schermo). Semplici simboli grafici, chiamati «icone», rappresentano oggetti d'ufficio fami-



Il mondo immaginario di viti e fiori è visto dall'interno di questa immagine al calcolatore creata da Ned Greene del New York Institute of Technology. I tralci delle viti disegnano gli spigoli di un reticolo tridimensionale, analogo alla struttura cristallina del diamante. L'immagine è un fotogramma che proviene da una sequenza animata nella quale il punto di vista si muove lungo uno dei «corridoi» formato dai tralci di vite. Gli oggetti della scena sono stati prima di tutto definiti matematicamente come reti di poligoni. Le viti sono state rese con una tecnica di mappatura particolare che dà l'impressione di rilievo, regolando l'ombreggiatura in base all'informazione di profondità ottenuta da immagini a raggi X di un modello in gesso di una corteccia reale di albero. Le foglie e i petali dei fiori sono stati colorati cartografando «dipinti», precedentemente registrati, sulle loro rappresentazioni a maglie con una tecnica «a tessitura». Per produrre nei petali una gradazione continua di colore, sono stati assegnati dei colori ai vertici delle rappresentazioni a maglie dei petali, e i colori dei vertici sono stati poi interpolati attraverso le facce poligonali del programma di restituzione. L'effetto della nebbia è stato ottenuto riducendo il contrasto con una funzione esponenziale di distanza dallo spettatore. «Senza la nebbia o qualsiasi altra cosa che dia il senso di profondità - nota Greene - la scena risulta praticamente incomprensibile.» Vi sono circa 1,9 milioni di poligoni nella scena e la loro restituzione visiva ha richiesto oltre 18 ore macchina di un minicalcolatore VAX 11/780 della Digital. Ogni pixel dello schema di scansione *(raster)* 1536 × 1536 porta 24 bit di informazione di colore. Alla stesura dei programmi hanno partecipato, oltre a Greene, Jules Bloomenthal, Paul Heckbert e Lance Williams.



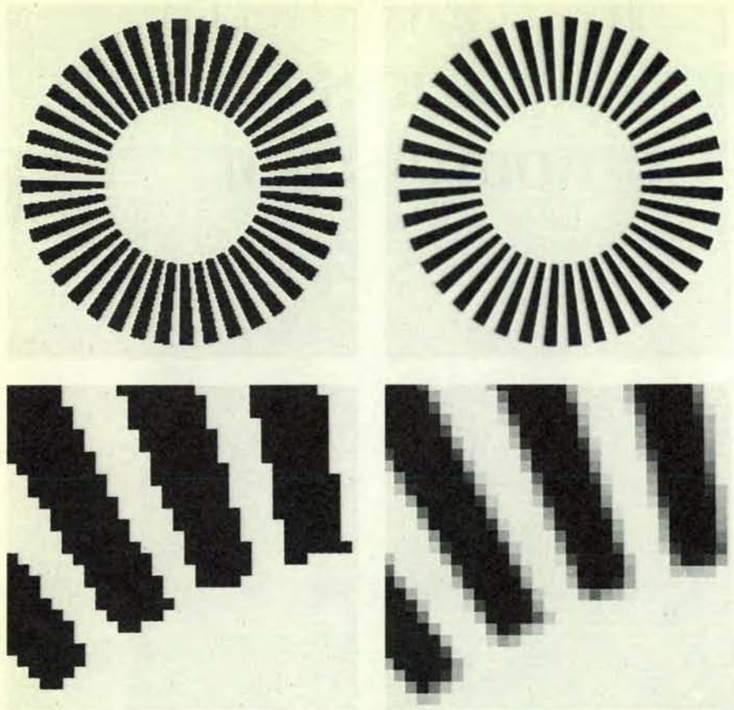
Esistono due modi operativi per produrre un'immagine sullo schermo di un tubo a raggi catodici. In un sistema a vettori il fascio di elettroni viene continuamente guidato fra due punti qualsiasi dello schermo in modo da creare una linea retta chiamata vettore. Il semplice disegno al tratto di una casa, per esempio, è il risultato di molte di queste operazioni. In un sistema a scansione, il fascio traccia uno schema regolare tipo quello televisivo di righe orizzontali di scansione e l'intensità del fascio viene aumentata nei pixel più vicini alle rette per formare la figura. La grafica a scansione è diventata ultimamente la forma dominante di visualizzazione al calcolatore.

liari, come cassetti, cartelle, cestini per la carta, calcolatrici e orologi; le funzioni che essi simboleggiano possono venire scelte indicandoli. Si è trovato che un'interfaccia basata su menù e icone è preferita dalla maggior parte degli utenti a un'interfaccia strettamente alfanumerica perché, quando questi simboli grafici sono opportunamente raffigurati, sembrano più naturali, sono più facili da apprendere e da usare, richiedono minor memorizzazione e portano a meno errori.

La grafica al calcolatore è diventata comune anche in molti altri contesti della vita quotidiana. I bambini (e anche molti adulti) stanno facendosi una cultura grafica con i videogiochi e svolgendo esercizi educativi basati su effetti visivi che spesso comportano una buona dose di animazione e interazione. Inoltre vi sono artisti che producono avvincenti e spettacolari animazioni al calcolatore per pubblicità televisiva e per effetti speciali nei film di fantascienza, utilizzando grandi quantità di tempo macchina per produrre ogni singolo fotogramma con gradi elevatissimi di dettaglio. La sintesi delle immagini, attualmente uno dei settori in maggior espansione della grafica al calcolatore, verrà analizzata in maggior dettaglio nel seguito.

La maggior parte degli schermi per la grafica interattiva si basa sulla tecnologia del tubo a raggi catodici (sebbene visualizzatori piatti a stato solido stiano entrando nell'uso comune, per esempio per i calcolatori portatili). In un tubo a raggi catodici il fascio di elettroni colpisce uno schermo rivestito di fosfòri, che emette luce con un'intensità dipendente dall'energia cinetica degli elettroni. Poiché l'emissione di luce dei fosfòri si affievolisce in millesimi di secondo, l'intera immagine deve essere ridisegnata a intervalli frequenti, normalmente 30 volte per secondo o più; la ripetizione del disegno si basa su una rappresentazione digitale dell'immagine immagazzinata in un unità di memoria chiamata memoria di rinfresco *(refresh buffer)*.

Il fascio di elettroni è guidato verso il punto desiderato sullo schermo in due modi: il modo a vettori o il modo a scansione sistematica orizzontale. In uno schermo a vettori il fascio può essere deflesso in modo continuo fra ogni coppia di punti del sistema di coordinate *(x, y)* bidimensionale dello schermo, in modo da creare una nitida linea retta chiamata vettore. Il risultato di parecchie di queste operazioni è un disegno al tratto. Anche i caratteri sono composti da brevi vettori. Un insieme di «primitive» di base - linee, archi, caratteri e altri elementi di immagine - è immagazzinato nella memoria di rinfresco sotto forma di una lista di comandi in codice, specificanti le coordinate degli estremi e altri attributi delle primitive come spessore, intensità e colore. Per i sistemi con capacità di visualizzare in tempo reale immagini tridimensionali, è fornito un hardware speciale per compiere l'operazione di «trasformazione di vista» consistente nel proiettare le primiti-



In un sistema di visualizzazione a scansione, lungo le linee o i bordi che non sono né orizzontali, né verticali, appaiono delle «seghettature» dovute al fatto che le «primitive» di questo tipo di visualizzazione sono approssimate da insiemi discreti di pixel vicini. Questo artefatto si nota a malapena nella configurazione radiale a sinistra in alto mentre si vede chiaramente nell'ingrandimento a sinistra in basso. Un modo per minimizzare il problema in sistemi in cui a ogni pixel siano fatti corrispondere più bit è quello di variare l'intensità dei pixel che si trovano ai bordi, in modo da sfumarli come nella configurazione a destra in alto e nel corrispondente ingrandimento a destra in basso. Il problema dei bordi seghettati è un problema di disturbo, o rumore (*«aliasing»*); la soluzione mostrata qui è chiamata *«anti-aliasing»*. Le immagini sono state prodotte sullo schermo di un calcolatore da Paul S. Strauss e James K. Rinzler della Brown University.

ve tridimensionali sullo schermo bidimensionale.

Mentre l'immagine viene continuamente «rinfrescata», il calcolatore stesso o hardware *ad hoc* può essere controllato per assegnare i valori di traslazione, rotazione o di scala agli estremi dei vettori o ai parametri di trasformazione di vista in modo da cambiare l'immagine per il successivo ciclo di rinfresco. Questi parametri possono essere specificati da un programma di animazione o da un operatore, usando un mouse, un *joystick* o un quadrante. La capacità di far muovere in modo continuo sullo schermo gli oggetti o il punto di vista dell'utente si è rivelata di grande aiuto per fornire alle persone la percezione del movimento quando esplorano la struttura di uno scenario tridimensionale poco familiare.

La grafica a vettori, che inizialmente è stata la più diffusa nella visualizzazione al calcolatore, offre diversi vantaggi: rappresenta le primitive in maniera da richiedere poca memoria; le primitive sono disegnate nitidamente e l'operatore può cambiare continuamente immagine in tempo reale. Il suo principale svantaggio è che non può mostrare superfici continue; sia gli oggetti bidimensionali sia quelli tridimensionali devono essere rappresentati da diagrammi a reticolo. Inoltre, se sullo schermo vi sono troppe primitive perché sia possibile ridisegnarle tutte nel tempo assegnato a un singolo ciclo di rinfresco, vi sono troppo pochi cicli e l'immagine sfarfalla.

Nella visualizzazione a scansione orizzontale il fascio non è deflesso secondo una configurazione determinata dall'immagine che va disegnata. Al contrario, come nel caso di un apparecchio televisivo, il fascio traccia una configurazione regolare di scansione. L'unico controllo riguarda l'intensità del fascio. In uno schermo a colori l'intensità dei tre fasci - uno per il rosso, uno per il verde e uno per il blu - è controllata singolarmente; ogni fascio colpisce il fosfòro corrispondente in una terna di punti rosso, verde e blu per ogni pixel. Le primitive in uno schermo a scansione vengono formate intensificando i pixel più vicini alla retta, alla curva o allo spigolo definiti dagli estremi delle primitive stesse. Le aree continue vengono riempite intensificando tutti i pixel interni. I visualizzatori a scansione orizzontale, a causa della loro configurazione a deflessione fissa, sono normalmente più semplici e meno costosi dei visualizzatori a vettori. D'altra parte, i primi richiedono una maggiore capacità nella memoria di rinfresco, che deve immagazzinare un valore di intensità o un valore di colore costituito come minimo da un bit per ogni pixel sullo schermo. In questo contesto la memoria di rinfresco è anche conosciuta come memoria di quadro *(frame buffer)* o mappa di bit. Un vantaggio dell'immagazzinamento di immagini sotto forma di singoli pixel, anziché di primitive di livello superiore, è costituito dal fatto che la prima rappresentazione risulta completamente indipendente dal numero delle primitive specificate per la visualizzazione; ne consegue che per gli schermi a scansione orizzontale non si presenta il problema dello sfarfallio delle immagini.

Una visualizzazione a vettori traccia linee e angoli in modo analogo a quello di un disegnatore che usa il righello e la squadra. Uno schermo a scansione, invece, si basa sulla versione elettronica della tecnica del «divisionismo» sviluppata nel XIX secolo dal pittore impressionista francese Georges Seurat. Una tecnica a campionatura discreta di questo genere può rendere evidenti i singoli pixel, e le primitive che non sono né orizzontali né verticali hanno margini seghettati. Questo artefatto è una forma di *«aliasing»*, ossia una forma di disturbo o rumore, un problema comune nell'elaborazione dei segnali. L'effetto può essere minimizzato aumentando la risoluzione del visualizzatore o variando l'intensità dei pixel che giacciono sui margini al fine di sfumarli. (Quest'ultima tecnica è chiamata talvolta *anti-aliasing*). Una nuova tecnica che simula maggiori risoluzioni ed evita la seghettatura è stata chiamata «messa in fase del pixel» *(pixel phasing)* dalla Megatek che l'ha sviluppata. Con questo metodo ogni pixel può essere leggermente riposizionato spostandolo di un quarto, di un mezzo o di tre quarti del diametro del pixel, in senso orizzontale o verticale; la dimensione del pixel può anche essere calibrata al fine di riempire gli spazi vuoti.

La capacità di specificare indipendentemente il valore di intensità o di colore di ciascun pixel in sistemi a scansione è particolarmente importante per la creazione di fonti di caratteri o di icone particolarmente dettagliate. Tipicamente una fonte è definita come un insieme di piccole matrici di pixel, una per ogni carattere o icona. Quando dobbiamo richiamare un carattere o un'icona, la sua matrice viene copiata dalla memoria centrale del calcolatore o da una parte speciale della memoria di quadro sulla regione appropriata dello schermo. D'altra parte, per il gran numero di pixel che vanno aggiornati quando una parte significativa dell'immagine viene spostata o cancellata, i cambiamenti sono abitualmente più lenti sugli schermi a scansione che non su quelli a vettori, sui quali devono

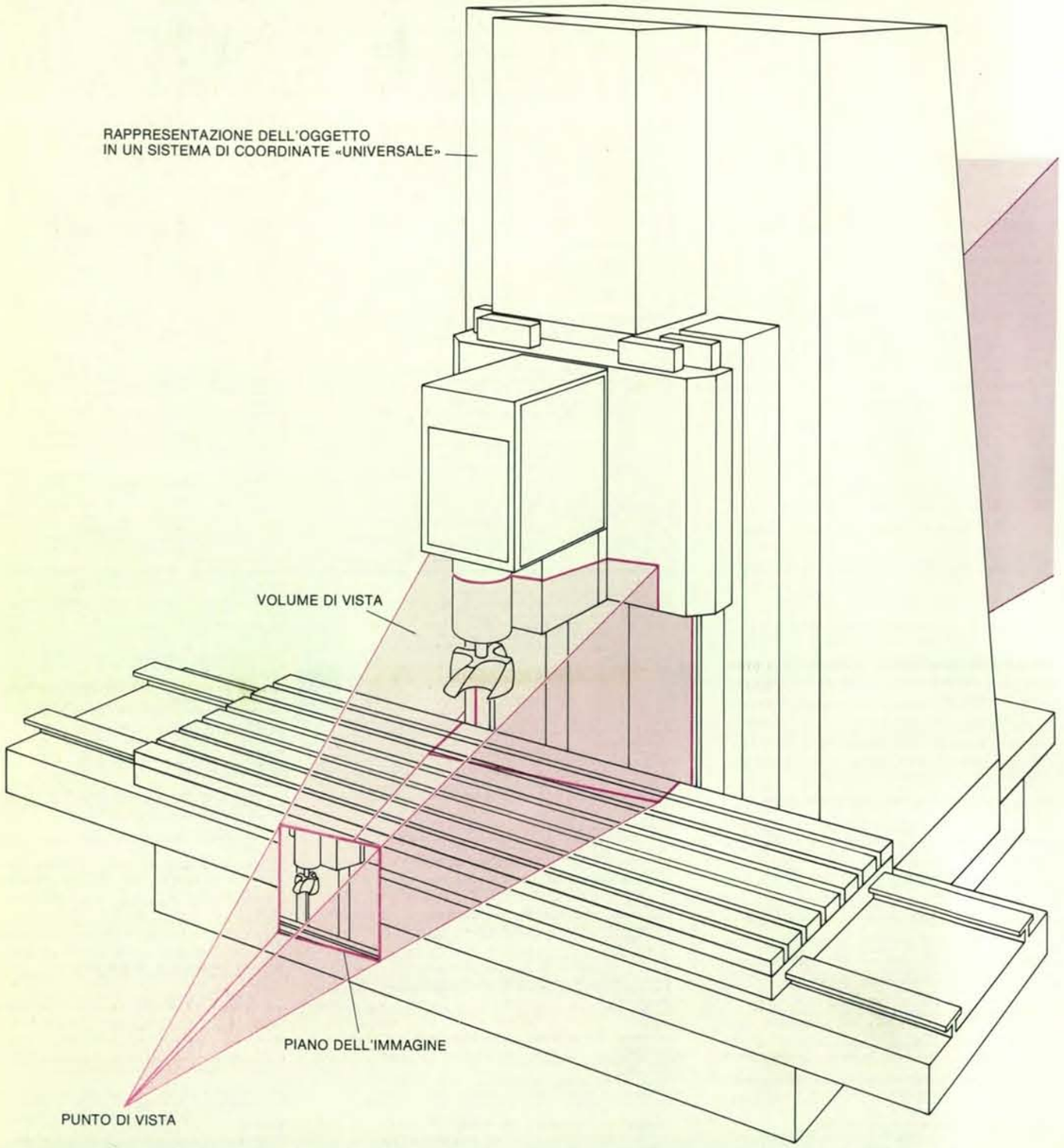
essere modificate solo le primitive codificate. I moderni sistemi a scansione orizzontale permettono al programmatore di copiare e muovere velocemente blocchi rettangolari nella memoria di quadro per mezzo di operazioni speciali che facilitano lo scorrimento del testo, la risistemazione delle finestre e la creazione di semplici sequenze animate. Tali sistemi possono inoltre fornire una rappresentazione a lista di visualizzazione, abbinata a una rapida trasformazione dalla forma di una primitiva codificata a quella di pixel per la memoria di quadro.

La funzione di un software di supporto di elevato livello è quella di isolare il programmatore da questi tipi di dettagli hardware di basso livello, in modo che si possa concentrare direttamente sull'applicazione. Agli albori della grafica al calcolatore ciò non era possibile, perché le applicazioni grafiche erano programmate a livello di linguaggio di assemblatore. L'efficienza era più importante della facilità della programmazione, e la possibilità di spostare i programmi da un modello di calcolatore a un altro difficilmente veniva presa in considerazione. Solo verso la fine degli anni sessanta e i primi anni settanta si iniziò a scrivere programmi di grafica a un livello elevato e a renderli indipendenti da sistemi particolari di calcolatori.

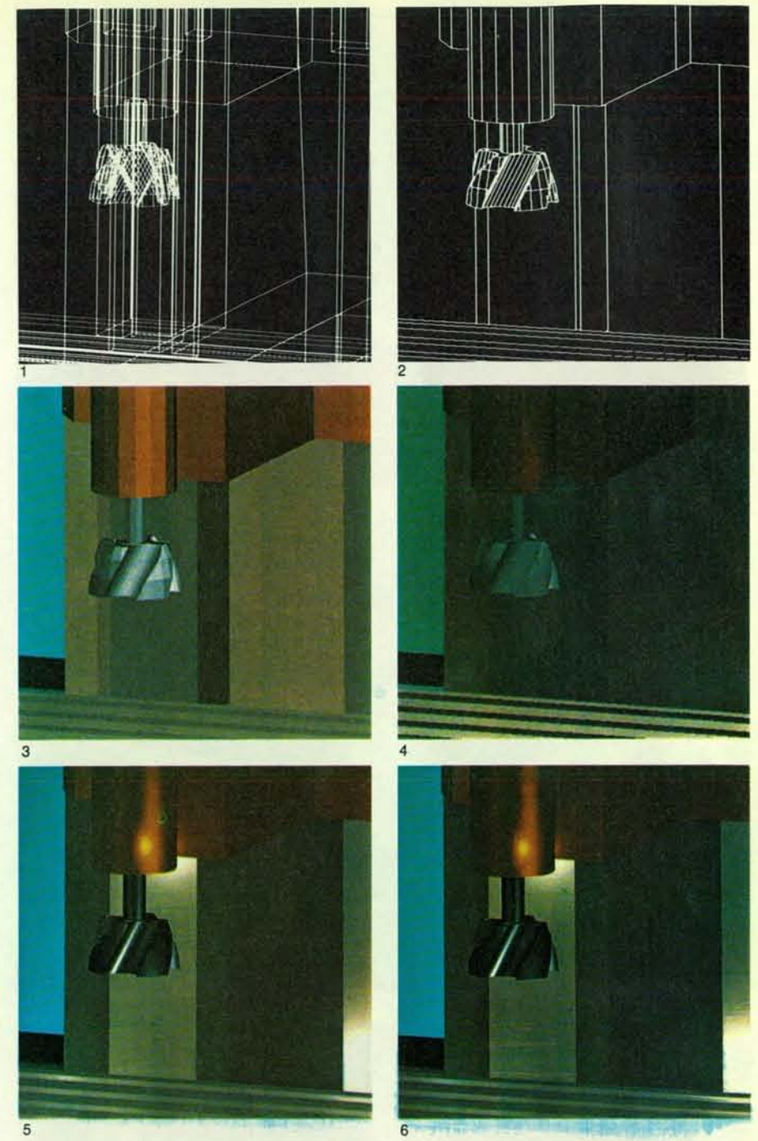
Il primo software di grafica della nuova era fu progettato su imitazione della strategia input-output dei linguaggi di programmazione di alto livello. Dispositivi «virtuali» ideali, corrispondenti a dispositivi reali interattivi vennero generati per mezzo di programmi di «guida del dispositivo» di basso livello, che manipolavano sia l'hardware per la grafica sia le comunicazioni input-output con l'unità centrale di elaborazione. Ogni visualizzatore virtuale aveva uno schermo virtuale quadrato progettato in modo da coincidere con il quadrato massimo che poteva adattarsi alla effettiva superficie di visualizzazione o a un *plotter* (tracciatore). Lo stesso sistema di coordinate unitario era utilizzato per indirizzare il visualizzatore virtuale indipendentemente dalle dimensioni dello schermo reale. Ogni visualizzatore virtuale poteva inoltre avere dispositivi virtuali di input. Dispositivi di input non disponibili su una particolare console potevano essere simulati per mezzo di altri dispositivi presenti; in questo modo si potevano creare, per esempio, una tastiera virtuale, un quadrante virtuale e perfino un mouse virtuale.

La maggior parte dei programmi per la grafica era sviluppata a quel tempo per applicazioni nella progettazione assistita dal calcolatore e nella visualizzazione di dati. I programmi giravano su sistemi di schermi a vettori e tracciavano figure derivate da una base di dati applicativi, nota come modello di applicazione. Il software per la grafica forniva al programmatore un sistema «universale» di coordinate (a due o tre dimensioni) idoneo a trattare nanometri, centimetri, chilometri o anni luce. Il sistema universale di coordinate permetteva al programmatore di astrarre la definizione di primitive a un livello ancor più lontano dall'hardware di quello del sistema di coordinate standard del visualizzatore virtuale. Il software manipolava inoltre l'intera operazione di trasformazione di vista, specificando la superficie volta a volta in vista nel sistema universale di coordinate e la regione del visualizzatore virtuale sul quale doveva apparire. Il software per la trasformazione di vista «tagliava» le primitive che si trovavano al di fuori dell'area di vista e proiettava sullo schermo reale solamente quelle che si trovavano all'interno dell'area di vista stessa. Nelle due dimensioni i confini di taglio corrispondevano a un rettangolo, mentre nelle tre dimensioni potevano essere o un parallelepipedo (nel caso di una proiezione parallela) o una piramide (nel caso di una proiezione prospettica).

In effetti, il tipo di software per la grafica sviluppato per primo oltre una decina d'anni fa può essere descritto dalla metafora della «cinepresa artificiale»: il programma applicativo costruisce un mondo che consiste di oggetti, quali simboli di un diagramma di flusso, elementi di un circuito o atomi nel modello dipendente dall'applicazione, includendo tutti gli attributi e i parametri connessi, e poi estrae le informazioni geometriche da passare al software per la grafica. Questo, che tipicamente è sotto il controllo dello spettatore, prende poi un'istantanea delle primitive specificate nel mondo dello spettatore, dal punto di vista specificato, e proietta l'istantanea sullo schermo. Del modello è dunque responsabile il programma applicativo e della visualizzazione di parte del modello è responsabile il



La trasformazione di vista è l'operazione nella quale una rappresentazione memorizzata di un oggetto (in questo caso una fresatrice idealizzata) è proiettata da un sistema di coordinate tridimensionale universale nel sistema di coordinate bidimensionale di uno schermo, rappresentato dal piano dell'immagine nel disegno. Imitando l'azione di una cinepresa, il software «taglia» le parti dell'oggetto fuori dal volume di vista, e poi proietta sullo schermo solo le parti che si trovano all'interno del volume. Nel caso di una proiezione prospettica tridimensionale, il contorno dei tagli è tipicamente una piramide. In questo caso gli spigoli nascosti sono stati eliminati prima dello stadio di proiezione.



La sintesi delle immagini al calcolatore può essere concepita come una sequenza di passi, in realtà spesso interconnessi in un unico programma. In questa dimostrazione del processo, l'oggetto (un primo piano di una fresatrice) è definito come una rete di poligoni ed è visualizzato anzitutto nella forma di un diagramma a reticolo *(1)*. Gli spigoli nascosti vengono poi eliminati *(2)*. Nel passo successivo l'ombreggiatura (in questo caso in colore) è applicata individualmente ai poligoni come una funzione dell'angolo tra il poligono e le sorgenti di luce e delle sue proprietà superficiali; il risultato è un'immagine di aspetto innaturale e sfaccettato *(3)*. Le discontinuità ai lati comuni tra due poligoni adiacenti possono essere eliminate con il metodo di ombreggiatura di Gouraud *(4)* e possono essere aggiunte zone di massima luce speculare con il metodo di ombreggiatura di Phong *(5)*. Nel passo finale l'*anti-aliasing* elimina le seghettature *(6)*. Le immagini sono state realizzate da Rinzler e Strauss in collaborazione con Roger L. Gould, Richard L. Hagy, David H. Laidlaw e Gerald I. Weil, studenti della Brown University.

L'animazione di alta qualità e in tempo reale è ottenuta in alcune applicazioni per scopi specifici, come si può vedere in questa sequenza di fotogrammi scelti da un sistema interattivo di simulazione di volo per l'addestramento di piloti al rifornimento in volo. Il sistema ad altissima prestazione, in grado di generare 50 fotogrammi per secondo, è prodotto dalla Evans & Sutherland.

software della cinepresa artificiale. L'applicazione stessa consta di due sottosistemi: il primo, un redattore grafico, che permette allo spettatore di creare e manipolare il modello di applicazione e la sua rappresentazione visiva, e un insieme indipendente di pacchetti di postelaborazione che analizza il modello di applicazione completato. Nella progettazione assistita dal calcolatore questi pacchetti includono mezzi per la simulazione e la verifica del progetto e successivamente per la specifica dei dati di fabbricazione e costruzione, operazioni eseguite spesso da macchine a controllo numerico.

Due pacchetti standard di grafica sono attualmente disponibili per tutte le categorie di visualizzatori commerciali: il Core Graphics System, tridimensionale, finanziato dalla Association for Computing Machinery, e il Graphical Kernel System, bidimensionale, adottato dalla International Standards Organization. Derivati da un predecessore comune, entrambi sono essenzialmente pacchetti di cinepresa artificiale. Come spesso succede, furono progettati molto tempo prima che la grafica a scansione diventasse la forma dominante di visualizzazione al calcolatore. Sebbene essi possano gestire primitive di scansione, come matrici di pixel e poligoni pieni, funzionano ancora nel sistema universale di coordinate con oggetti definiti dall'utente. Per molte applicazioni semplici della grafica a scansione, il programma di applicazione non può sfruttare appieno i vantaggi del pacchetto per giustificare il considerevole tempo di calcolo in più necessario per poter gestire applicazioni più complesse. Inoltre, un programma che si basa su un pacchetto di grafica può non essere in grado di sfruttare appieno una serie di nuove e potenti capacità di hardware, associate a stazioni di lavoro per grafica a scansione e a personal computer.

I programmi che non seguono lo schema basato sulla metafora della «cinepresa artificiale» includono i programmi di «pittura» che stanno diventando popolari nei sistemi di grafica a scansione. Gli oggetti manipolati in questi programmi non sono oggetti a coordinate universali, ma i singoli pixel, e il pacchetto deve permettere allo spettatore di ricolorare, spostare o perfino combinare logicamente regioni arbitrarie nella memoria di quadro. Dipingere manipolando pixel nella memoria di quadro è analogo a fare una ripresa fotografica alterando le aree direttamente sull'emulsione, anziché esporre la pellicola attraverso una cinepresa puntata su una scena reale. I pacchetti di cinepresa artificiale non sono adatti per tali operazioni, di basso livello, dipendenti dal dispositivo.

La situazione si complica ulteriormente per la necessità di gestire finestre multiple su uno schermo a mappa di bit. I pacchetti grafici esistenti non hanno modo di gestire contemporaneamente più programmi applicativi. La maggior parte delle stazioni di lavoro con grafica a scansione è pertanto provvista di un «gestore di finestre», una parte di basso livello del software del sistema, che tiene traccia di quale programma giri in una finestra e di dove la finestra sia definita sullo schermo. Una finestra potrebbe far girare un programma di pittura, un'altra un programma di elaborazione di testi e una terza un programma applicativo basato su un pacchetto standard di grafica. Il gestore di finestre deve trattare problemi come spostamenti, coperture e scoperture di finestre che si sovrappongono, operazioni di scala e di taglio delle primitive per adattarle alla parte visibile di una finestra e infine la trasformazione nella versione a scansione delle primitive visibili per mostrarle sullo schermo. Finora non vi sono progetti comunemente accettati per tali gestori di finestre.

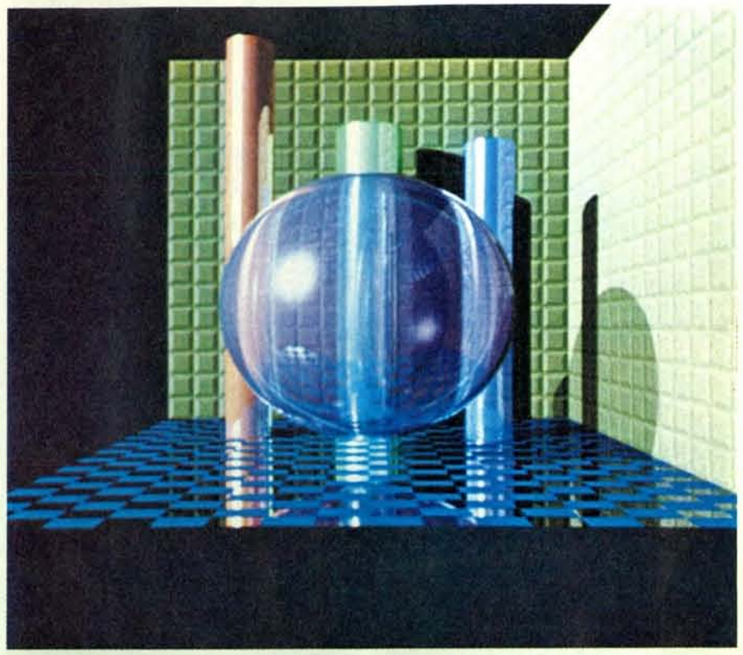
Per quanto riguarda l'immediato futuro, coesisteranno numerosi standard grafici progettati da diverse organizzazioni di utenti. Come esempi si possono citare l'Initial Graphics Exchange Specification, uno standard per il disegno tecnico applicato alla progettazione assistita dal calcolatore e la North American Presentation Level Protocol Syntax, per la visualizzazione di testi e di grafici in televisione. Tutti gli standard condividono l'intenzione comune di definire le primitive, i loro attributi e i loro raggruppamenti in collezioni ben definite per poterle manipolare selettivamente. Alla fine tutti questi standard diversi dovrebbero essere unificati.

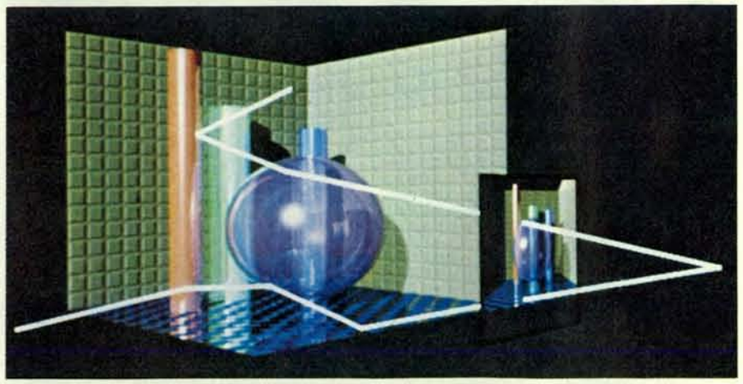
La maggior parte delle applicazioni tradizionali della grafica al calcolatore si è avuta nel campo bidimensionale. Di recente però è aumentato l'interesse commerciale per le applicazioni tridimensionali, derivante dal notevole progresso, fatto negli ultimi dieci anni, sul duplice problema di costruire modelli di scene tridimensionali e di visualizzarle il più realisticamente possibile. Per esempio, nei simulatori di volo per l'addestramento dei piloti, l'enfasi è posta sulla risposta all'input sia del pilota sia dell'istruttore. Per dare l'idea di movimento continuo, il simulatore deve presentare immagini pressoché realistiche di un panorama la cui dinamica varia con una frequenza di almeno 30 fotogrammi al secondo. A differenza di questa animazione in tempo reale, le immagini pubblicitarie e cinematografiche vengono elaborate fuori linea, spesso per ore, al fine di ottenere il massimo di realismo o di efficacia visiva. Nella progettazione assistita dal calcolatore c'è la tendenza a creare interattivamente grafici a reticolo e poi visualizzarli subito in versione finita. L'hardware di più recente produzione rende perfino possibile la creazione interattiva di oggetti poliedrici «solidi».

I modelli di oggetti bidimensionali sono costruiti da primitive come rette definite da due estremi, poligoni definiti da una lista di vertici e possibilmente da uno schema di riempimento, cerchi definiti da un centro, un raggio e possibilmente da uno schema di riempimento, e curve polinomiali definite dai loro coefficienti. In tre dimensioni le primitive corrispondenti vengono definite aggiungendo la coordinata z. Si possono definire anche primitive che esistono solo nelle tre dimensioni come poliedri, piramidi, sfere, cilindri e superfici descritte da certe funzioni polinomiali.

Sistemi di costruzione di modelli di solidi per creare oggetti tridimensionali si basano sulla specificazione di parametri o interattiva o fuori linea. La specificazione fuori linea può venir fatta con archivi di dati creati da un altro programma o con un redattore di testi. In alternativa può essere utilizzata una descrizione procedurale, come quella usata per generare curve e paesaggi frattali. Inoltre si può dare il modello di un oggetto direttamente come un solido, o indirettamente come un volume limitato dalla sua superficie.

Nei sistemi basati sulla geometria solida costruttiva, i modelli degli oggetti vengono realizzati direttamente per mezzo di





La tecnica a tracciamento di raggio si basa su un algoritmo che richiede una grande quantità di tempo per calcolare la riflessione e la rifrazione della luce da parte di superfici di oggetti immaginari. Fondamentalmente il programma di calcolatore traccia singoli raggi di luce, cominciando dal punto di vista e passando all'indietro attraverso ogni pixel nel piano dell'immagine, finché il raggio colpisce una superficie. Si continua poi a vedere il raggio riflesso come se provenisse da una sorgente di luce, o direttamente o dopo la sua riflessione da parte di un altro oggetto. Per una superficie trasparente va tracciato un secondo raggio rifratto. In questa illustrazione della tecnica, prodotta da Lee Westover e Turner Whitted della Università del North Carolina e della Numerical Design Ltd., in alto è mostrata un'immagine tracciata a raggio mentre la tecnica con la quale è stata ottenuta l'immagine è mostrata in basso nel diagramma generato dal calcolatore. Le linee bianche tracciano due raggi riflessi; le componenti rifratte sono omesse.

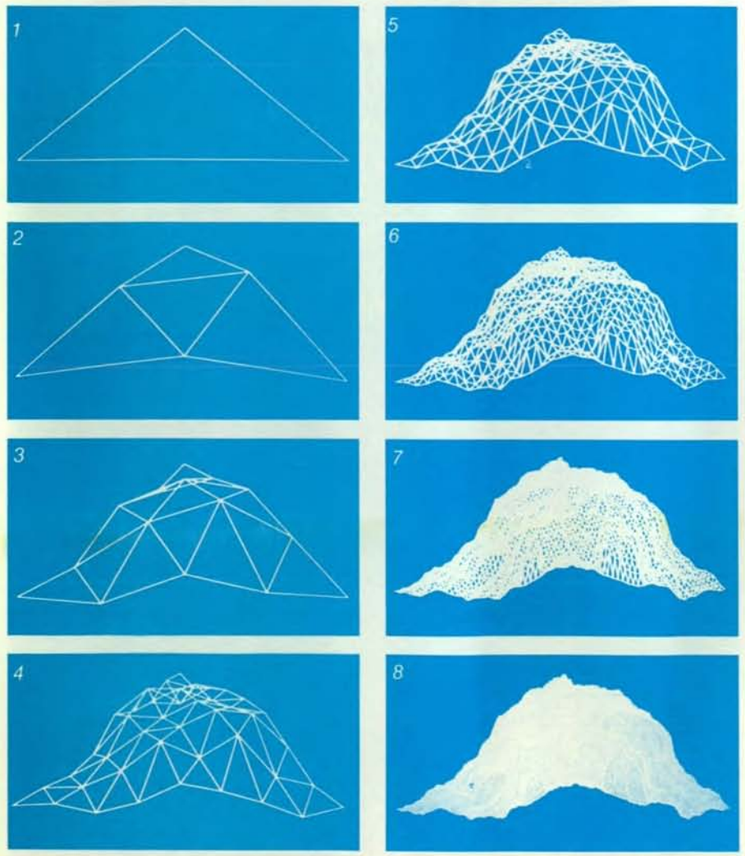
primitive solide, come parallelepipedi, cilindri e sfere. Si possono combinare le primitive per mezzo di operazioni su insiemi tridimensionali, quali l'unione (congiungendo due oggetti), l'intersezione (prendendo un sottoinsieme in comune) e la differenza (prendendo tutto del primo oggetto eccetto quelle parti che ha in comune con il secondo). La rappresentazione indiretta viene fatta in sistemi di rappresentazione di confini che possono anche fornire operatori di insiemi, ma che definiscono un oggetto come contornato da sfaccettature poligonali, cilindriche, o perfino da zone di superficie definite da funzioni polinomiali. Tale definizione di superficie «a forma libera» con zone curve è importante per le industrie automobilistiche e aerospaziali ai fini di definire la forma dei loro veicoli.

Un oggetto dotato di simmetria rotazionale può anche essere descritto da una superficie di rivoluzione; un vaso o una bottiglia è definito dalla sua generatrice (la curva del suo profilo) e da un asse di rivoluzione. Analoga alla generazione per rotazione è la generazione per traslazione: una faccia di complessità arbitraria, anche con buchi, viene traslata lungo una curva spaziale per creare un volume. Un ingranaggio ipotetico può essere fatto definendo dapprima un quarto di sezione di una faccia, la quale può essere completata con operazioni di simmetria, e quindi rigenerando la faccia lungo un breve percorso rettilineo al fine di definire la forma cilindrica dell'ingranaggio solido.

Molte altre tecniche matematiche sono utili per definire classi di oggetti e sistemi ibridi includono molte tecniche diverse. Il caso particolare della creazione interattiva di oggetti presenta con questi metodi un problema ulteriore, in quanto l'utente è obbligato a guardare una proiezione bidimensionale di una scena tridimensionale, di cui è difficile valutare la profondità. Fra le tecniche che forniscono all'utente alcune verifiche al procedere del processo di specificazione vi sono le viste multiple (come le comuni proiezioni ortografiche frontali, laterali e superiori, e anche una vista prospettica tridimensionale), il disegno in un piano di coordinate x, y o z costanti, e ausilii come linee di quota aggiornate dinamicamente e griglie bi- o tridimensionali opportunamente segnalate.

Quando gli oggetti nella scena sono stati definiti, la fase successiva consiste nel passare la descrizione dell'oggetto ai programmi di sintesi dell'immagine per l'operazione di restituzione visiva. Gli attuali algoritmi per la sintesi delle immagini lavorano o con descrizioni poligonali, o con definizioni polinomiali o di ordine superiore di superfici matematiche. È comune ridurre definizioni di livello superiore a una più semplice approssimazione «spezzata», con una rete di piccoli poligoni antecedente all'operazione di restituzione visiva. Il processo di restituzione può essere idealizzato come una sequenza di passi che spesso però sono interconnessi in un programma reale. Tutti questi passi sono fondamentalmente degli adattamenti delle leggi fondamentali dell'ottica.

Il primo passo consiste nell'eliminazione delle superfici nascoste, cioè superfici o parti di superfici che non sono visibili dal punto di vista della cinepresa artificiale. Questa categoria include sia le superfici dei lati opposti sia quelle coperte da altri oggetti più vicini al punto di vista. Varie tecniche per l'eliminazione delle superfici nascoste possono essere realizzate via hardware. Gli algoritmi tipicamente partono dal presupposto che lo schermo si trovi nel piano di proiezione $z = 0$ per la scena che si trova dietro di esso. Per esempio, l'algoritmo *z-buffer* genera una memoria separata (chiamata per l'appunto *z-buffer*) di valori z, un valore per ogni pixel. Il valore z di un pixel registra la profondità a cui si trova il punto corrispondente sul poligono più vicino incontrato fino a quel momento e che si proietta sul pixel. Quando un nuovo poligono è trasformato, tagliato e proiettato sul piano $z = 0$, i valori z dei suoi pixel vengono confrontati uno alla volta con quelli immagazzinati nello z-buffer. Un pixel del poligono è «in vista», e viene di conseguenza immagazzinato sia nella memoria di rinfresco sia nello z-buffer, solo se il suo valore z è minore di quello immagazzinato in quel momento nello z-buffer (il che significa che in corrispon-



Questa tecnica semplice per generare un'immagine realistica di una montagna si basa grosso modo sui concetti di geometria frattale originariamente formulati da Benoit B. Mandelbrot del Thomas J. Watson Research Center dell'IBM. La dimostrazione della tecnica è riprodotta per gentile concessione della Lucasfilm Ltd. Partendo dal triangolo singolo mostrato nel passo *1*, il programma del calcolatore genera il passo *2* con la procedura seguente. Anzitutto, suddivide ogni lato del triangolo nel punto centrale. In secondo luogo sposta ogni punto centrale di una distanza proporzionale alla lunghezza del lato corrispondente. (Il fattore di proporzionalità può essere generato a caso o preso da una tavola, per esempio, di 100 numeri casuali ben distribuiti.) In terzo luogo collega i tre nuovi punti a un altro punto per formare quattro nuovo triangoli. Il passo *3* è generato dal *2* usando la stessa procedura a turno per ognuno dei quattro nuovi triangoli, generando 16 triangoli, a ognuno dei quali si applica ancora la procedura nel passo *4*, e così via. Sebbene l'algoritmo di suddivisione sia semplice, può portare a una superficie poligonale molto complessa. La superficie a forma di montagna, nel passo *8*, può poi essere restituita visivamente con tecniche convenzionali di grafica al calcolatore per produrre un paesaggio finito.

L'immagine composita di un paesaggio costiero, intitolata «Point Reyes», è stata prodotta da un gruppo di operatori della Lucasfilm. Il paesaggio è stato definito con molte tecniche diverse; i vari elementi della scena sono stati restituiti visivamente uno per volta e successivamente combinati. La semplice tecnica di costruzione di modelli mostrata nell'illustrazione di pagina 100 è stata impiegata da Loren Carpenter per definire le rocce, i monti e i laghi; lo stesso Loren ha redatto inoltre il programma per le superfici nascoste e un programma di «atmosfera» per il cielo e la foschia. Rob Cook ha diretto la produzione, ha progettato la strada, le colline, lo steccato e l'arcobaleno e ha scritto il software per la mappa della tessitura. Tom Porter ha fornito la tessitura disegnata proceduralmente per le colline e ha redatto il software per combinare gli elementi in una immagine composita. Bill Reeves ha definito l'erba per mezzo di un proprio sistema di «particelle in movimento» e ha redatto il software per la costruzione dei modelli. David Salesin ha posto le increspature sulle pozzanghere; le piante fiorite sono opera di Alvy Ray Smith.

denza di quel pixel il poligono è più vicino allo schermo di tutti i poligoni incontrati in precedenza). Un pixel di un poligono dunque diventa visibile solo se non è stato sostituito quando viene elaborato l'ultimo poligono.

Mentre l'algoritmo *z*-buffer prende poligoni in un ordine qualsiasi, l'algoritmo del «pittore» prima di tutto li mette in ordine a partire dal fondo in avanti. Nel caso in cui coppie di poligoni non possano essere ordinati in modo semplice, essi vengono suddivisi finché è possibile. Poi vengono proiettati e «dipinti» nella memoria di rinfresco in ordine, dal fondo in avanti, in modo che i poligoni più vicini al punto di vista coprano i più distanti, senza bisogno di un ulteriore calcolo.

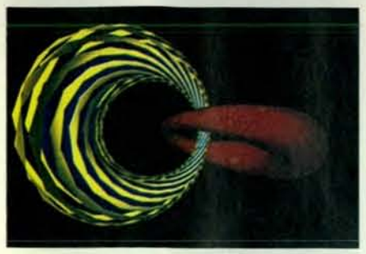
Dopo aver calcolato le superfici visibili, si passa a calcolare l'ombreggiatura di ciascuna. La regola di ombreggiatura deve tener presenti le proprietà della superficie (colore, fattore di riflessione, tessitura), nonché posizione, orientazione e proprietà relative di altre superfici e delle fonti di luce. Modelli di illuminazione per fonti di luce possono tenere conto della luce dell'ambiente, di sorgenti puntiformi (come il Sole o una lampadina a incandescenza) o di sorgenti distribuite (come una finestra, o una fila di tubi fluorescenti).

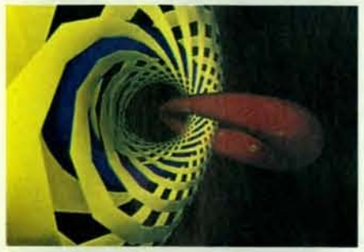
Per la luce d'ambiente la costruzione di modelli è più facile, se si aggiunge una quantità di intensità di luce costante a tutte le superfici, ma ovviamente questa strategia non offre alcun modo per differenziare le varie superfici. Il modello del riflesso di sorgenti puntiformi di luce da parte di superfici opache (quelle che diffondono la luce egualmente in tutte le direzioni) è costruito secondo la legge di Lambert, in base alla quale l'intensità varia secondo il coseno dell'angolo fra la direzione della sorgente di luce e un vettore perpendicolare alla superficie, chiamato normale alla superficie. La massima illuminazione si ha quando la superficie è perpendicolare alla sorgente di luce. Per superfici più brillanti che producono zone di massima luce speculare come nel caso di legno o metalli lucidati, la quantità di luce riflessa dipende sia dall'angolo della sorgente luminosa sia dall'angolo del punto di vista rispetto alla normale alla superficie. La superficie agisce come uno specchio in quanto riflette la maggior parte della luce solo quando gli angoli sono quasi uguali (cioè quando il punto di vista e la sorgente luminosa sono posti simmetricamente rispetto alla normale alla superficie). Più gli angoli differiscono, più rapidamente l'intensità della luce decresce. Aggiungendo le componenti della luce ambientale la riflessione speculare e diffusa dà l'intensità di una singola superficie. Nel caso vi sia colore, vi è un'equazione per ognuno dei tre colori fondamentali.

L'effetto di questa combinazione di operazioni ha un aspetto innaturale e sfaccettato. Dato che il poligono viene descritto da una singola normale a una superficie, i poligoni adiacenti con normali alla superficie differenti hanno diversi valori di intensità, e vi è una notevole discontinuità nel lato condiviso. Il metodo di ombreggiatura di Gouraud (dal suo inventore, Henri Gouraud) media i valori di intensità ai vertici dei poligoni e poi attraverso le linee di scansione, per ottenere continuità. Il metodo di Phong (dal nome del suo inventore Bui-Tuong Phong) migliora l'ombreggiatura di Gouraud utilizzando un calcolo molto più dettagliato che è più sensibile agli effetti direzionali delle zone di massima luce speculari: i più moderni sistemi di grafica a scansione di medio prezzo e di elevate prestazioni sono ora in grado di svolgere l'intero lavoro di restituzione visiva per circa 3000 poligoni al secondo. Essi procedono dapprima elaborando una gerarchia di oggetti, inclusa l'applicazione delle trasformazioni geometriche per simulare il moto, poi calcolando la trasformazione di vista e successivamente eseguendo gli algoritmi relativi alle superfici nascoste e all'ombreggiatura continua. Alcuni anni fa questo livello di prestazione era disponibile solo su simulatori di volo del costo di milioni di dollari.

Altri effetti da trattare sono le ombre, la trasmissione della luce e le proprietà della superficie, come tessitura e grana. Gli algoritmi relativi alle ombre per sorgenti puntiformi assomigliano agli algoritmi per l'eliminazione di superfici nascoste, in quanto stabiliscono quali superfici possono essere «viste» dalle sorgenti di luce. Le superfici che sono visibili simultaneamente dal punto di vista e dalle sorgenti di luce non sono in ombra, mentre lo sono quelle visibili dal punto di vista, ma non dalla sorgente di luce. Per sorgenti di luce distribuita, i calcoli complessivi devono includere sia l'ombra sia la penombra.

La trasmissione della luce è un argomento ancora più difficile. La trasmissione «speculare», caratteristica di superfici trasparenti come il vetro, è determinata dall'indice di rifrazione della sostanza. La trasmissione diffusa attraverso materiali traslucidi, come vetro smerigliato, provoca una diffusione in tutte le direzioni. Gli algoritmi più complessi e più realistici per trattare dal punto di vista computazionale sia la riflessione sia la rifrazione sono chiamati «a tracciamento di raggio». In sostanza, essi tracciano singoli raggi di luce per determinare quale di essi arriva al punto di vista e come arriva. Per evitare di dover trattare con un'infinità di linee uscenti da una sorgente puntiforme, il processo lavora al contrario, a partire da ogni pixel. Ogni raggio che comincia dal punto di vista e che passa attraverso un pixel viene proiettato all'indietro finché colpisce una superficie. Il tracciamento all'indietro del raggio riflesso continua per determinare se proviene da una sorgente di luce o dalla riflessione di un altro oggetto. Per una superficie trasparente va anche tracciato un secondo raggio rifrat-





L'applicazione matematica della grafica al calcolatore è rappresentata in questa sequenza di fotogrammi, adattata dal film animato al calcolatore *Topology and Mechanics* di Huseyin Kocak, Frederick Bisshopp, Thomas F. Banchoff e David Laidlaw della Brown University. Un'ipersfera, un analogo quadridimensionale di una sfera ordinaria, può essere visualizzata riempiendola con due cerchi interconnessi e una successione di superfici circostanti toroidali. (L'operazione è pressoché analoga a quella di tagliare una sfera in due punti ai poli opposti e una serie di cerchi paralleli tra di essi.) I fotogrammi mostrano una serie di proiezioni prospettiche su uno spazio tridimensionale, fatte da un punto di vista sulla ipersfera, di due superfici toroidali (una in blu e una in rosso), che avvolgono strettamente i due cerchi dell'ipersfera. Una terza superficie toroidale (in giallo) viene mostrata mentre si muove dalla superficie blu alla superficie rossa in sei passaggi. La superficie gialla è stata tagliata in bande per rivelare il suo collegamento con le altre due superfici. Questo metodo di «scomposizione» di una ipersfera deriva da un lavoro del 1931 del matematico tedesco Heinz Hopf. Il calcolatore è inestimabile per la realizzazione di procedure matematiche di questo tipo, perché può manipolare facilmente oggetti astratti in uno spazio di dimensione superiore.

to. In effetti ogni raggio è una sonda di cui si deve controllare l'intersezione con ogni oggetto; in ultima analisi solo una piccola percentuale di raggi ha antecedenti in una sorgente di luce. Esistono nuove tecniche per trattare la riflessione e la rifrazione di luce diffusa, ma sono ancora molto costose in termini di quantità di calcolo richiesta.

La tessitura superficiale può essere trattata da vari modelli che incorporano irregolarità locali. Per disegnare una figura bidimensionale su una superficie, si può usare uno schema di valori di intensità per modulare le intensità calcolate dagli algoritmi relativi alle ombre e all'ombreggiatura. Alcuni dei più recenti lavori nella sintesi di immagini riguardano effetti come profondità di campo, sfocatura da movimento e restituzione realistica di oggetti che in natura presentano sia regolarità sia irregolarità statistiche quali montagne, acqua, cielo, alberi e cespugli.

Sia che si abbia a che fare con semplici diagrammi a blocchi o con rappresentazioni altamente realistiche, la funzione più importante della grafica al calcolatore è quella di aumentare la comprensione delle persone, di metterle in grado di sperimentare senza pericolo, senza angoscia o costi aggiuntivi e di aiutarle a rispondere a domande del tipo «Ma se...». Per la maggior parte degli studi relativi alla costruzione di modelli e alla simulazione non sono sufficienti però rappresentazioni statiche: i fenomeni che normalmente si vogliono comprendere sono dinamici. Una immagine statica può valere un migliaio di parole, ma una sequenza animata vale spesso molte immagini statiche. Una delle capacità chiave della nuova generazione di potenti stazioni di lavoro è quella di rivelare il comportamento degli oggetti mentre mutano nel tempo attraverso l'animazione in tempo reale controllata dall'utente.

Fra gli oggetti che mostrano un comportamento dinamico e che sono di particolare interesse per i programmatori, ci sono i programmi e le loro strutture di dati. Sin dall'inizio della grafica al calcolatore nei primi anni sessanta vi è sempre stato un considerevole interesse nei confronti della progettazione diagrammatica dell'hardware e del software. Diagrammi a blocchi, diagrammi di flusso e di interconnessione di moduli, e molte altre rappresentazioni simboliche sono stati usati per raffigurare sistemi i cui progetti erano specificati con la stesura di istruzioni in linguaggio testuale. Sebbene oggi larga parte della progettazione dell'hardware sia svolta con simboli grafici, non vi sono ancora linguaggi di programmazione nell'uso comune in cui gli elementi base siano pittorici e non testuali.

I programmi di grafica sono dunque specificati in un linguaggio di programmazione ordinario, come il FORTRAN o il Pascal con «chiamate» per pacchetti grafici *ad hoc*, non specificando graficamente che cosa si vuole. Le ragioni di questo curioso contrasto fra le tecniche di specificazione per l'hardware e quelle per il software includono la compattezza e la precisione che offre un linguaggio di programmazione convenzionale e la facilità con la quale si possono fare cambiamenti con un redattore di testi che offra buoni strumenti per la ricerca di specifici gruppi di parole o di caratteri. Manca comunque l'esperienza con veri linguaggi pittorici.

Sebbene gli scienziati dei calcolatori conoscano piuttosto bene l'attrezzatura di cui un programmatore ha bisogno per specificare *come* fare qualcosa in un linguaggio di programmazione convenzionale, non sono ancora in grado di gestire il problema più complesso di permettere a un utente di specificare *che cosa* vada fatto e poi far compilare automaticamente al sistema una procedura appropriata a partire da questa specifica. Di pari passo si sono fatti progressi nello sviluppo di ambienti di programmazione basati su stazioni di lavoro. Generalmente queste attrezzature permettono al programmatore di scrivere interattivamente e di eliminare gli errori dai programmi con il supporto di viste multiple di programmi e di dati, rappresentati in forma di testo o di icone. Le viste vengono aggiornate dinamicamente man mano che si esegue il programma.

Due aree promettenti per l'applicazione di una grafica al calcolatore dinamica sono le aule e i laboratori. Molte scuole e università si stanno indirizzando all'insegnamento basato sul microcalcolatore. Una delle tecniche adottate è lo stile tradizionale di «istruzione programmata» assistita dal calcolatore, la quale pone l'accento sull'acquisizione di fatti e di capacità. L'istruzione assistita dal calcolatore sta ora migliorando grazie alla sperimentazione simulata di laboratorio e ad «ambienti di consultazione», nei quali le informazioni sono disponibili come in un'enciclopedia o in una biblioteca. Cinque anni fa, per iniziativa del mio collega Robert Sedgewick, il Dipartimento di scienza dei calcolatori della Brown University iniziò a studiare le applicazioni della tecnologia delle stazioni di lavoro all'insegnamento e alla ricerca. L'anno scorso inaugurammo una nuova «classe elettronica», attrezzata con 55 stazioni di lavoro a elevate prestazioni, connesse da una rete ad alta velocità. La maggior parte dei corsi introduttivi di informatica viene impartita in questo auditorium predisposto *ad hoc*, in quanto vi si tengono corsi in equazioni differenziali, geometria differenziale e neurologia.

L'aula elettronica sviluppata da Robert Sedgewick e dall'autore, insieme ai colleghi del Dipartimento di scienza dei calcolatori della Brown University, è attrezzata con 55 stazioni di lavoro a prestazione elevata, collegate da una rete ad alta velocità. L'istruttore prima può illustrare un argomento servendosi di una sequenza animata di immagini, vista da tutti gli studenti, e poi lasciare che ciascuno lavori indipendentemente sul medesimo «filmato interattivo». L'aula costruita *ad hoc* è stata usata per corsi di scienza dei calcolatori, di matematica e di neurologia.

Il nostro scopo è quello di dare agli studenti l'opportunità di «vedere» un fenomeno astratto e quindi di sviluppare in proposito intuizioni geometriche prima di entrare nei particolari della programmazione e della matematica. Intendiamo anche coinvolgerli nella pratica più rapidamente, combinando la lezione in aula all'esperimento di laboratorio. La maggior parte degli studenti è passiva durante una lezione formale, anche quando vengono stimolate delle domande. Ne consegue che non verificano realmente quanto appreso finché non svolgono compiti a casa e/o pratica di laboratorio. Ora un istruttore può introdurre un nuovo argomento parlando attraverso una sequenza animata di immagini viste da tutti gli studenti e poi lasciando che ognuno lavori indipendentemente sullo stesso «filmato interattivo».

Il nostro supporto fondamentale, il Brown Algorithm Simulator and Animator (BALSA) permette agli utenti di controllare la velocità di una sequenza animata, di decidere quali viste del soggetto guardare e di specificare quali dati di input elaborare. Esiste anche la possibilità di programmi che vanno all'indietro e «disfano» i loro effetti grafici.

Viste dinamiche multiple di oggetti complessi si sono rivelate utili non solo a studenti ma anche a ricercatori. È stato avviato molto lavoro alla Brown University per imparare come generalizzare la tecnica, sia ad altri argomenti scientifici e tecnologici sia a campi privi di una tradizione di rappresentazione grafica dei propri oggetti e metodi di studio. Sono queste le ragioni intrinseche per l'assenza di una rappresentazione pittorica in alcuni campi, oppure è un mero artificio culturale? Se si trovano molti altri modi di usare la grafica in svariate discipline, come potrebbe cambiare la pedagogia in aula? Questa domanda è di notevole importanza, dato che le stazioni di lavoro cominciano a moltiplicarsi negli ambienti universitari e molti studenti vi possono accedere 24 ore su 24 direttamente dalle loro stanze. Come possono docenti non abituati alla programmazione specificare e sviluppare materiale didattico, senza dover investire nella preparazione per l'insegnamento assistito dal calcolatore 100 o più ore (come usualmente avviene) per ogni ora d'aula? Molti anni di ricerca e di sperimentazione saranno necessari prima che questa promettente nuova capacità possa essere davvero largamente diffusa.

Le nostre dimostrazioni interattive tenute in aula alla Brown University sono un esempio della più ampia categoria dei cosiddetti «libri elettronici». Un altro esempio assai differente è lo Spatial Data Management System, sviluppato al Massachusetts Institute of Technology. Questo sistema rende possibile consultare in un «paese di dati» bidimensionale una scrivania arbitrariamente grande popolata da icone. Il cursore può essere mosso per puntare a un'icona qualsiasi ed esporne i contenuti. La base di dati archivia testi, diagrammi, fotografie, suoni e fotogrammi televisivi (recuperati in tempo reale da un video disco).

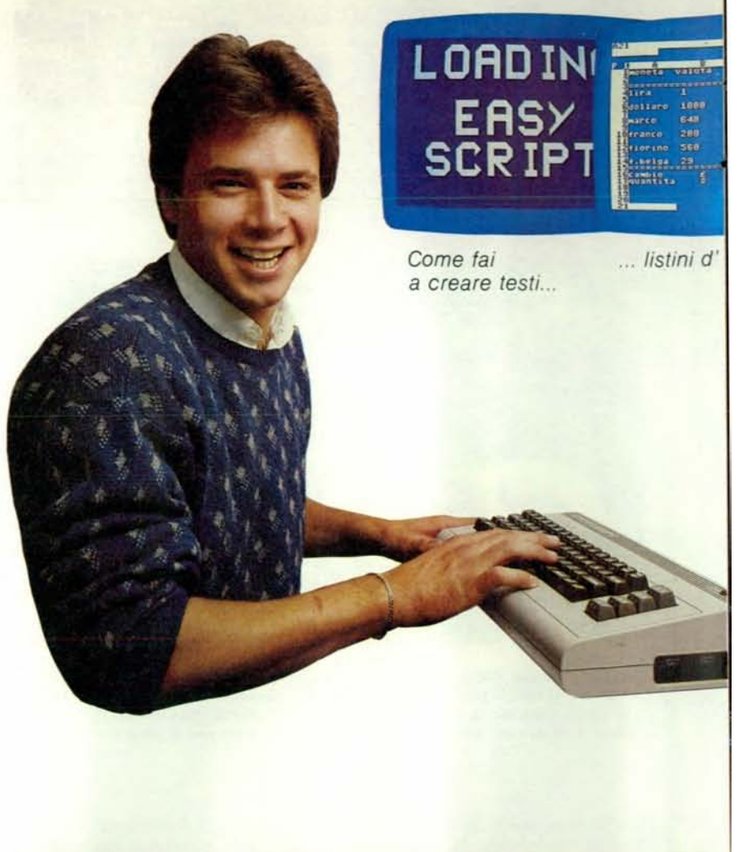
In un sistema sperimentale analogo sviluppato alla Brown University, la metafora della scrivania viene sostituita da quel-

la di una rete associativa di pagine, di note in calce, di note a margine e di rimandi incrociati. Il lettore può seguire un percorso di rimandi incrociati tra argomenti affini come si farebbe con un'enciclopedia. Un manoscritto non lineare di questo tipo è stato chiamato «ipertesto» da Theodor H. Nelson, un *leader* del movimento per lo sfruttamento della tecnologia della visualizzazione al calcolatore per creare una nuova forma letteraria. Altri progetti alla frontiera della grafica al calcolatore, iniziati da Nicholas P. Negroponte e collaboratori al MIT, includono un manuale interattivo per la manutenzione e la riparazione delle automobili, in grado di spiegare il testo creando appositi «film» basati su sequenze di fotogrammi prelevate da videodischi, e un giornale elettronico che continuamente analizza i notiziari delle agenzie di stampa e la sua base di dati e crea resoconti e immagini per il lettore sulla base di un profilo dei suoi interessi memorizzato.

I professionisti del calcolatore, e in particolare gli specialisti della grafica, vedono finalmente coronate le loro aspettative sulla utilità e la comodità dell'informatica per il pubblico generico. La grafica al calcolatore, una volta dominio dei soli esperti, è ormai un fatto comune, se persino i bambini delle elementari usano finestre, mouse e visualizzazioni al calcolatore come strumenti per disegnare e, in effetti, immaginare. Il pensare e il programmare in termini grafici stanno diventando parte integrante dell'apprendimento per la costruzione di algoritmi.

E che dire del futuro? Vi dovrà essere ancora un miglioramento di ordini di grandezza nel rapporto prezzo-prestazioni dell'hardware prima che l'utente medio possa avere l'equivalente di un simulatore di volo in tempo reale sul proprio tavolo, e un ulteriore progresso dovrà esser fatto nella comprensione dell'interazione delle leggi della fisica e dell'estetica, prima che gli artisti del calcolatore siano in grado di rendere in modo convincente scene realistiche che appaghino anche l'occhio. Infine veri visualizzatori tridimensionali apriranno le porte di un nuovo regno: la ricerca su ologrammi digitali può permettere la elaborazione in tempo reale di scene verosimili. Dal lato dell'input è necessario ulteriore lavoro sull'interfaccia verso l'utente. Per esempio, le immagini devono essere meglio integrate con il suono, come nel caso dell'input e dell'output della voce. Si dovranno fare anche molti progressi nella comprensione della struttura del linguaggio naturale e nelle aree collegate dell'intelligenza artificiale, prima che gli utenti possano conversare con i loro calcolatori. Sono anche necessari nuovi metodi per fornire controllo tattile e retroazione nell'esplorazione delle «sensazioni» degli oggetti visti sullo schermo. Per quanto mi riguarda l'ideale ultimo è efficacemente espresso nel vecchio fumetto «Mandrake il Mago»: «Mandrake gesticola ipnoticamente...», e nel luccichio di un occhio è evocata una nuova scena, un nuovo ambiente sensoriale.

COMMODORE 64,



Come fai a creare testi...

... listini d'

Software per la gestione dell'informazione

Memorizzare grandi quantità di dati è utile solo se le informazioni sono recuperabili rapidamente e in forma comprensibile: i programmi debbono riflettere la struttura della base di dati e quella fisica della memoria

di Michael Lesk

Chiunque lavori nello scompiglio di un ufficio ingombro sa che non basta avere molte informazioni a portata di mano per raggiungere prontamente quella che serve di volta in volta. Negli ultimi vent'anni la capacità delle macchine elettroniche di memorizzare informazioni è cresciuta rapidamente e il costo di questa operazione è calato in modo altrettanto rapido. In genere, tuttavia, il software per organizzare e recuperare i dati memorizzati per via elettronica non ha tenuto il passo e chi si dedica alla stesura dei programmi per la gestione delle informazioni si trova nella necessità di rincorrere le capacità crescenti degli elaboratori elettronici.

Quali sono i principi che guidano questo sforzo? Uno è quello secondo cui la forma migliore di organizzazione dipende dal contenuto delle informazioni e da come dovranno essere usate. Per esempio, i programmi che amministrano un elenco di nomi vengono scritti per molti scopi diversi; questi programmi differiscono quindi di molto, a seconda delle informazioni associate a ciascun nome dell'elenco e di come i nomi vengono recuperati. In commercio si trova un sistema chiamato Soundex, che viene impiegato per individuare i viaggiatori che abbiano prenotato un dato volo. Il Soundex registra i nomi in base a un criterio fonetico, il che riduce il numero degli errori di trascrizione e consente di ritrovare un nome anche se non se ne conosce la grafia esatta. Il Chemical Abstracts Service si serve di alcuni programmi per stabilire se una certa sostanza è già stata identificata, funzione che sotto il profilo formale è analoga a quella espletata dal Soundex. I sistemi tuttavia non sono fonetici e per di più registrano parecchie informazioni sulla struttura chimica e sulla nomenclatura. Se da una parte il ricorso a informazioni specifiche di una disciplina può aumentare l'efficacia di un programma nell'adempimento di un certo lavoro, d'altra parte rende il programma meno adatto ad altri scopi. I programmi studiati per una base di dati chimici non funzionerebbero altrettanto bene nella gestione dell'elenco dei passeggeri su voli di linea.

Un altro fattore da tener presente è la struttura fisica del dispositivo di memoria. Verso la fine degli anni settanta si diffuse l'impiego dei dischi magnetici per memorizzare i dati. Su un disco, i dati sono registrati in sottounità chiamate settori e la massima efficienza d'accesso si ottiene quando le frontiere logiche fra i dati più o meno coincidono con i confini tra i settori. Quindi il software allestito per gestire grandi quantità di informazioni è vincolato da una parte alla struttura delle apparecchiature e dall'altra al contenuto delle informazioni. I tentativi di progettare un software che sia in grado di sfruttare più a fondo la capacità delle macchine attuali devono sostanzialmente iscriversi nei limiti fissati da questi due vincoli fondamentali.

Un gruppetto di elementi fra loro collegati in un sistema elettronico per la memorizzazione dei dati prende di solito il nome di *record*. Per esempio, in un archivio che descriva l'assortimento dei prodotti in vendita in un supermercato, ciascun record potrebbe comprendere il tipo di prodotto, la categoria generale dei prodotti di cui fa parte, il numero del reparto in cui lo si trova e il prezzo. Ciascun elemento del record, ad esempio il tipo di prodotto, viene detto campo. Il record viene rintracciato nell'archivio elettronico per mezzo di una chiave, cioè di un contrassegno che può consistere in un campo, in una parte di campo o in una combinazione di più campi.

Certi tipi di campi vengono impiegati come chiavi più spesso di altri. È probabile che, nell'esempio della base di dati del supermercato, il numero del reparto possa servire da chiave, ma il prezzo no. In tal caso l'utente potrebbe risalire agevolmente ai prodotti in vendita nel reparto 3, ma non potrebbe rintracciare agevolmente tutti i prodotti che costano 500 lire. Altre informazioni, per esempio il nome del grossista che ha fornito il prodotto, quanto quest'ultimo rimane sullo scaffale e la scorta in magazzino, possono o meno essere usate come chiavi. I programmi usati per gestire le informazioni dovrebbero rendere facile la ricerca di quel record che contiene un determinato valore della chiave.

Si osservi che non è necessario che la chiave venga ricavata direttamente dal record. Quando fui incaricato di allestire

Gli itinerari più brevi da Wall Street all'Hotel Plaza, a New York, sono stati calcolati con un programma che ha tracciato anche la pianta della porzione meridionale dell'Isola di Manhattan, riprodotta nella pagina a fronte. Un itinerario *(linea in bianco)* corrisponde al tempo minimo di spostamento e l'altro itinerario *(linea in arancione)* alla distanza minima da percorrere. A causa della disposizione dei sensi unici a Manhattan bassa, ogni itinerario comincia con un cambiamento di direzione. L'itinerario che rende minimo il tempo passa lungo West Street e giunge fino alla Tenth Avenue, che è seguita in direzione dei quartieri alti. Sono le informazioni memorizzate relative al tempo medio di trasferimento per ciascuna via che consentono al programma di individuare l'itinerario che richiede il tempo minimo. L'itinerario che copre la distanza minima comincia dalla West Street, poi passa lungo la Sixth Avenue, sempre in direzione dei quartieri alti. I programmi per memorizzare ed elaborare le informazioni contenute nella pianta sono stati ideati dall'autore e dai suoi collaboratori. Il programma per individuare gli itinerari che corrispondono al tempo minimo o alla distanza minima è stato scritto da uno di questi collaboratori, Jane Elliott, che lavora presso i Laboratori AT&T Bell.

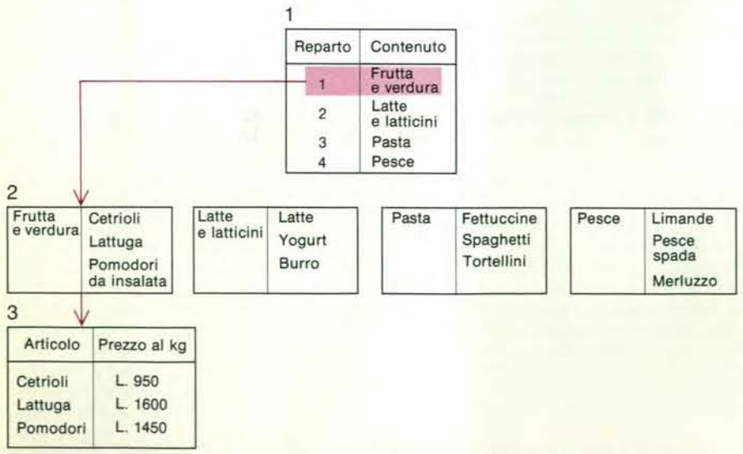
un sistema di assistenza per l'elenco telefonico dei Laboratori AT&T Bell, per costruire la chiave trasformai i nomignoli contenuti nel record in memoria nei loro equivalenti formali: così quello che nel record era «Chuck» nella chiave diventò «Charles». Quella trasformazione, tuttavia, era valida soltanto per gli elenchi telefonici: in una base di dati geografici la città di Billings, nello stato del Montana, non dovrebbe certo essere trasformata in Williamings.

Tanta varietà di relazioni tra record, campi e chiavi serve a definire i tre criteri principali secondo cui organizzare i record elettronici: quello gerarchico, quello a rete e quello relazionale. Il sistema gerarchico prende il nome dall'ordinamento esistente fra i campi del record. In ciascun gruppo di record uno dei campi viene assunto come campo principale, mentre gli altri campi sono subordinati a quello. I gruppi di record vengono disposti in un ordine seriale che ricorda i pioli di una scala e i dati possono essere recuperati solo se si attraversano i livelli secondo un percorso definito dalla successione dei campi principali.

Le basi di dati gerarchiche sono state usate fin dagli inizi dell'era del calcolo automatico moderno, negli anni quaranta, ed esempi che ne illustrano il funzionamento si possono incontrare in ambiti svariati. Per fare un esempio elementare, consideriamo la base di dati del supermercato, che abbiamo già descritto. Il primo livello di organizzazione in un corpus di informazioni siffatto potrebbe comprendere una tabella che desse i numeri dei reparti e la categoria generale degli articoli in vendita in ciascun reparto. La categoria degli articoli serve da campo principale. Solo dopo aver rintracciato la tabella dei reparti e del loro contenuto e dopo avere scelto una categoria, per esempio quella della frutta e verdura, si possono raggiungere le tabelle del livello successivo.

Il secondo livello della gerarchia potrà comprendere tabelle in cui siano elencati i singoli prodotti venduti in un reparto. Le tabelle di livello inferiore potranno contenere il prezzo di ciascun prodotto, il fornitore all'ingrosso e la durata del prodotto sullo scaffale. Solo percorrendo uno dopo l'altro vari livelli della gerarchia si può ottenere un'informazione su un articolo particolare, per esempio il suo prezzo. A meno che entro l'archivio non vengano inseriti indici suppletivi, è dispendioso, in termini di risorse del calcolatore, porre domande che si discostino dal percorso gerarchico, per esempio chiedere direttamente il prezzo dell'articolo.

Il modello a rete è alquanto più flessibile di quello gerarchico, perché in esso si possono istituire tra gli archivi molteplici collegamenti, i quali permettono all'utente di accedere a un dato archivio senza dover percorrere tutta la trafila gerarchica che lo precede. Quindi i collegamenti sussidiari modificano in maniera rilevante la struttura verticale della base di dati. Per esempio, nel caso del supermercato si potrebbe istituire un collegamento tra l'elenco dei reparti e la tabella dei prezzi: così sarebbe possibile trovare il prezzo di un articolo senza dover prima reperire la tabella intermedia che contiene i prodotti in vendita in quel reparto.



Una base di dati gerarchica consiste in tabelle che debbono essere esplorate in un ordine stabilito a priori per poterne recuperare le informazioni. L'illustrazione mostra i passi da compiere per trovare il prezzo di un prodotto in un archivio che contenga dati sull'assortimento dei prodotti nel magazzino di un supermercato. La prima tabella della gerarchia individua i reparti e la categoria generale dei prodotti in vendita in ciascun reparto *(1)*. La categoria generale può essere impiegata per reperire una tabella al livello successivo, che elenchi i prodotti allineati sugli scaffali del reparto *(2)*. La terza tabella comprende il prezzo di ciascun prodotto *(3)*. Questo tipo di organizzazione sarebbe comodo per un impiegato che dovesse apporre il prezzo su ogni articolo, perché il metodo di accesso segue il percorso dell'impiegato attraverso il supermercato. Sarebbe invece meno comodo per rispondere direttamente alle domande dei clienti sui prezzi. In un tipo di organizzazione strettamente connesso a questo e che si chiama base di dati a rete, si potrebbe introdurre un collegamento sussidiario fra la prima e la terza tabella. Questo collegamento ridurrebbe il tempo necessario per trovare il prezzo, ma richiederebbe altro spazio in memoria.

Il modello relazionale, che fu proposto verso il 1970 da E. F. Codd, dell'International Business Machines Corporation (IBM), è attualmente oggetto di grande interesse, poiché sembra assicurare una flessibilità maggiore degli altri tipi di basi di dati. Sia nell'organizzazione gerarchica sia in quella a rete, certe domande ottengono una risposta molto più rapida che altre; in più le domande alle quali è difficile e le domande alle quali è facile rispondere sono determinate nel momento in cui la base di dati viene costituita e in molti casi non c'è alcun criterio fondato per individuare a priori le domande che saranno poste più spesso.

Nella base di dati relazionale la flessibilità viene conseguita con l'abolizione della gerarchia tra i campi: tutti i campi possono essere usati come chiavi per ottenere informazioni. Un record non viene concepito come un insieme di entità discrete, tra le quali un elemento venga considerato il campo principale: al contrario, ciascun record è considerato come una riga di una tabella bidimensionale e ciascun campo è una colonna della stessa tabella.

L'elemento

Reparto 2 Latte e latticini
Latte 960 lire/litro

può essere considerato come una relazione tra il campo del reparto, il campo della categoria generale, il campo del prodotto e il campo del prezzo. La relazione potrebbe essere ampliata con il nome del grossista, la durata del prodotto, la scorta di magazzino e altre indicazioni. Relazioni più brevi, composte di due soli campi, si possono ottenere dall'insieme completo delle relazioni scegliendo i campi giusti; questo procedimento si chiama proiezione della relazione. Quindi il prezzo di qualsiasi prodotto può essere recuperato rapidamente, e così pure una tabella in cui siano elencati tutti i prodotti presenti in un reparto.

Quando si parla di gestione delle informazioni, si fa di solito riferimento alla distinzione tra le forme di organizzazione gerarchica, a rete e relazionale delle basi di dati; ma questo schema di classificazione non è utile quanto potrebbe sembrare, poiché la struttura di qualunque base di dati può essere arricchita tramite indici secondari che consentono di rispondere in modo efficiente anche a quesiti che non si conformino con l'organizzazione soggiacente. Inoltre certi problemi sono comuni a tutti e tre i tipi di basi di dati: si consideri il compito di scegliere un record in un archivio composto da molti record tra loro collegati. L'archivio potrebbe consistere di voci di un dizionario, righe di una guida telefonica o altri record, ma in ogni caso il problema è quello di risalire a un record in modo rapido ed efficiente.

Quando si voglia estrarre un singolo record, un parametro critico è il modo in cui gli elementi sono disposti nella memoria principale del calcolatore o nelle sue memorie secondarie, per esempio i dischi magnetici. Vi sono numerose tecniche che permettono di disporre i dati in modo da agevolarne in seguito il reperimento; queste tecniche si possono adottare per tutti e tre i tipi di basi di dati.

Se il corpus delle informazioni è piccolo, può non essere necessario adottare un assetto particolare. Le righe possono essere registrate su disco secondo una successione arbitraria; quando si deve reperire un certo elemento, un semplice programma che rileva la coincidenza delle configurazioni esamina le righe una dopo l'altra finché non compaiano determinate combinazioni di simboli. Nel sistema operativo Unix, per esempio, esiste un programma, chiamato *grep*, che viene spesso adoperato in questo modo.

Supponiamo che in un archivio chiamato *telnos* sia stato memorizzato un elenco di numeri telefonici, fra i quali anche quelli di due ditte fornitrici di acqua di seltz. Il comando *'grep seltzer < telnos* dell'Unix fa sì che vengano stampate due righe che comprendono i nomi e i numeri delle ditte che distribuiscono seltz. Nell'archivio *telnos* la parola *seltzer* era stata immessa prima del nome della ditta, ma affinché il programma *grep* funzioni non è necessario che la chiave sia posta al principio della riga. Il comando *grep beverage < telnos* dà lo stesso risultato, perché la parola *beverage* fa parte del nome di entrambe le ditte.

La gestione di un archivio i cui elementi non siano posti in un ordine prefissato presenta parecchi vantaggi. Intanto si risparmia tempo e fatica quando si costituisce l'archivio, perché non è necessario ordinare i dati. Inoltre si acquista una notevole flessibilità, perché non è necessario decidere in anticipo quali elementi dell'archivio fungeranno da chiavi nel recupero dei dati. Se un utente del sistema decide di trovare tutti i numeri dell'archivio *telnos* che hanno 800 come prefisso, il comando Unix *grep 800 < telnos* ha effetto immediato, a prescindere dalla circostanza che questa richiesta fosse stata prevista o no quando l'archivio era stato allestito. Se la lista non è ordinata, è possibile aggiungere nuovi dati in coda all'elenco senza dover riorganizzare gli elementi già registrati. Inoltre non è richiesto più spazio di memoria di quello occupato dai singoli elementi, mentre nei sistemi di registrazione più complessi una parte della memoria viene occupata dall'informazione occorrente per organizzare i dati.

Il metodo di memorizzazione non ordinato soffre di uno svantaggio capitale: il recupero è molto lento. Ciascuna riga dell'archivio deve essere esaminata singolarmente e di conseguenza l'esplorazione di un archivio di 10 000 righe richiede un tempo cento volte superiore rispetto all'esplorazione di un archivio di 100 righe. L'algoritmo impiegato dal programma *grep* corrisponde al cercare un libro in una biblioteca cominciando dagli scaffali prossimi all'ingresso ed esaminando ciascun volume finché non si trovi quello desiderato.

Per accelerare il corso della ricerca all'archivio può essere dato un ordine seriale, per esempio l'ordine alfabetico o l'ordine numerico. Se gli elementi sono disposti in ordine seriale, è possibile individuarne uno qualunque ricorrendo alla tecnica chiamata ricerca binaria: l'elenco viene diviso a metà e il programma determina in quale metà è contenuto l'elemento cercato. Il processo viene poi ripetuto finché l'elemento è individuato.

Supponiamo che l'archivio consista nei lemmi di un dizionario e che si voglia cercare la definizione di «gatto». Un programma per la ricerca binaria individuerebbe per prima cosa il lemma centrale del dizionario (mettiamo che sia «legalità»). Confrontando la lettera iniziale di «legalità» con quella di «gatto», il programma stabilisce che «gatto» sta nella prima metà dell'archivio. Il lemma centrale della prima metà sia per esempio «distorto»: allora «gatto» sta nel secondo quarto del dizionario. Continuando a questo modo, mediante una successione di punti medi, tra i quali ci sono «geometria», «fattore», «formaggio» e «galassia», si riesce a individuare la definizione di «gatto».

La ricerca binaria è più veloce dell'esplorazione di un insieme non ordinato di dati. Supponiamo che l'archivio in cui si deve compiere la ricerca contenga n elementi. Se la ricerca viene compiuta nell'insieme non ordinato, l'individuazione di un elemento generico comporta in media $n/2$ operazioni. La ricerca binaria, invece, per individuare un dato elemento comporta circa $\log_2 n$ operazioni (dove $\log_2 n$ è il logaritmo in base 2 di n). L'ordinamento seriale ha un vantaggio ulteriore: quando un elemento è stato trovato, è semplice scoprire informazioni sugli elementi adiacenti, per esempio la parola che segue «gatto» nel dizionario.

Tuttavia l'aggiunta di un elemento nuovo a un archivio organizzato per la ricerca binaria, è un procedimento dispendioso, poiché l'ordinamento progressivo dell'archivio dev'essere conservato. Dato che l'elemento nuovo viene inserito, in media, a metà dell'elenco, ogni volta che viene aggiunto un elemento ulteriore metà degli elementi dell'archivio debbono essere spostati. Anche la circostanza che gli elementi possono essere reperiti solo tramite la chiave in base alla quale l'insieme è stato ordinato può costituire una seria limitazione. È certo possibile creare più copie degli elementi e memorizzare parecchi archivi, ciascuno ordinato in base a una chiave diversa, ma ciò porta a un notevole dispendio di spazio.

Un'altra tecnica per recuperare gli

1

Prodotto	Reparto	Categoria	Prezzo (Lire)	Unità
Burro	2	Latte e latticini	750	etto
Merluzzo	4	Pesce	1000	etto
Cetrioli	1	Frutta e verdura	950	chilo
Limande	4	Pesce	1550	etto
Lattuga	1	Frutta e verdura	1600	chilo
Fettuccine	3	Pasta	1230	mezzo chilo
Latte	2	Latte e latticini	990	litro
Spaghetti	3	Pasta	810	mezzo chilo
Pesce spada	4	Pesce	2500	etto
Pomodori da insalata	1	Frutta e verdura	1450	chilo
Tortellini	3	Pasta	385	etto
Yogurt	2	Latte e latticini	500	barattolo piccolo

2

Cetrioli	950	chilo

3

Cetrioli	Reparto 1	Frutta e verdura	950 al chilo	chilo
Lattuga	Reparto 1	Frutta e verdura	1600 al chilo	chilo
Pomodori da insalata	Reparto 1	Frutta e verdura	1450 al chilo	etto

Nella base di dati relazionale, ciascuna voce è costituita da un elenco di elementi collegati; è possibile recuperare rapidamente qualunque sottoinsieme degli elementi contenuti nell'elenco completo. Nella base di dati di un supermercato fra gli elementi ci possono essere, per esempio, il tipo di prodotto, il reparto dove si può trovare il prodotto, la categoria generale a cui esso appartiene e il suo prezzo *(1)*. Relazioni più particolari, come il prezzo di un articolo, si possono ricavare rapidamente dall'insieme completo delle relazioni *(2)*. Si può anche costruire una tabella in cui compaiano gli articoli esposti in un reparto e i loro prezzi *(3)*. La base di dati relazionale è vantaggiosa quando non si conosce in anticipo quali saranno le domande più frequenti.

elementi di un archivio ordinato è basata su sottogruppi di elementi adiacenti, chiamati «secchielli». Il primo elemento di ogni secchiello è registrato in una tabella che funge da indice delle suddivisioni dell'archivio. Se n elementi sono suddivisi in $\sqrt{n}$ secchielli, un'esplorazione lineare dell'indice dei secchielli, seguita da un'esplorazione del secchiello giusto per individuare l'elemento desiderato, comporta solo poco più di $\sqrt{n}$ operazioni. Benché il numero $\sqrt{n}$ non sia piccolo come $\log_2 n$, in particolare per un archivio esteso, non si tratta di un numero troppo ingombrante quando l'archivio è piccolo. Per di più, scrivere programmi per la memorizzazione a secchielli è facile.

Il numero minimo di operazioni che si è riusciti a conseguire finora per la ricerca in un archivio è $\log_2 n$. È possibile scendere al di sotto di questo limite tramite la procedura dell'indirizzamento casuale (*hashing*). Per comprendere i vantaggi di questo metodo, si pensi di assegnare un numero a ciascun elemento dell'archivio. Se ogni volta che si ha bisogno di un elemento fosse possibile calcolare rapidamente il suo numero per mezzo di un algoritmo semplice, l'elemento potrebbe essere richiamato direttamente dall'archivio senza bisogno di cercarlo.

Per i lemmi di un dizionario si potrebbe costruire una matrice con indici di tale fatta, in linea di principio con un metodo semplicissimo: a ciascuna lettera dell'alfabeto si può associare un numero, quindi la grafia di una parola individua uno e un solo numero, che serve da indirizzo della parola nella memoria. Il guaio di questo schema è che lo spazio di memoria occorrente è grandissimo e resterebbe quasi del tutto vuoto; infatti la maggior parte delle combinazioni di lettere non corrispondono ad alcuna parola della nostra lingua. Se vogliamo considerare il problema sotto un altro profilo, la difficoltà nasce dal fatto che le lettere iniziali delle parole non sono distribuite in modo uniforme. Ad esempio, le parole che cominciano per *c*, *s* o *a* sono molto più frequenti di quelle che cominciano per *u* oppure *z*. Quindi se 100 parole venissero associate a 100 indirizzi numerici sulla base delle loro iniziali, gli indirizzi si accumulerebbero in alcune caselle, invece di riempire tutte le caselle da 1 a 100 in modo uniforme.

La soluzione più comoda è costruire un algoritmo che assegni a ciascuna parola un numero «pseudocasuale». La grafia della parola determina univocamente il numero pseudocasuale, ma parole di grafia diversa possono generare lo stesso numero. L'algoritmo è basato su un'espressione matematica chiamata funzione di miscuglio (*hash function*). Una di queste funzioni assegna a ciascuna lettera dell'alfabeto un valore numerico e i valori di tutte le lettere della parola vengono poi sommati per fornire il numero pseudocasuale che serve da indirizzo. Se si sceglie una buona funzione di miscuglio gli elementi risultano distribuiti in modo abbastanza uniforme su tutta la matrice con indici, che in tal caso si chiama tabella d'indirizzamento casuale.

In genere è necessario lasciare vuoto almeno un quarto delle caselle della tabella d'indirizzamento casuale. Se alcune caselle sono lasciate vuote, si riduce la frequenza dei casi in cui lo stesso numero pseudocasuale viene assegnato a più di un elemento. Quando si presenta una duplicazione di questo genere, che viene chiamata collisione, si fa intervenire un algoritmo supplementare che assegni una casella al secondo elemento. Questo algoritmo potrebbe far riempire la casella successiva della tabella d'indirizzamento casuale; come alternativa potrebbe venir calcolata un'altra funzione di miscuglio. Se la distribuzione delle voci è nota (cosa che peraltro non capita spesso), la tabella può essere riempita per più di tre quarti.

Si osservi che, quando un corpus d'informazioni è strutturato in una tabella d'indirizzamento casuale, la funzione di miscuglio viene calcolata ogni volta che venga cercata una voce. Essa è in realtà facile da calcolare e, quando sia stata calcolata, l'elemento viene recuperato senza ulteriori ricerche. Quindi l'indirizzamento casuale è un metodo di recupero velocissimo. Anche l'aggiornamento dell'archivio è un'operazione molto pratica, perché gli elementi non sono memorizzati in un ordine seriale che debba essere conservato spostando molti elementi ogni volta che ne venga inserito uno nuovo.

Tuttavia, la tecnica dell'indirizzamento casuale semplice presenta un inconveniente che la rende poco adatta per molte applicazioni. Il problema delle collisioni fa sì che si debba conoscere in anticipo il numero approssimativo delle voci, in modo da poter costruire una tabella d'indirizzamento delle dimensioni giuste. Se inaspettatamente arrivano molti elementi nuovi, può essere necessario ricalcolare tutte le funzioni di miscuglio. In pratica, spesso non è possibile prevedere le dimensioni di un insieme di dati e quindi è fastidioso dover fissare in anticipo le dimensioni della tabella d'indirizzamento.

Gli archivi non ordinati, i secchielli, la ricerca binaria e l'indirizzamento casuale sono tutte tecniche oggi in uso, specie nel caso di basi di dati piccole, per le quali la facilità di programmazione è un parametro più importante che non l'efficienza complessiva. Se, invece, la base di dati è grande, si usano di preferenza due metodi scoperti negli ultimi anni: l'indirizzamento casuale estendibile e gli alberi *B*.

L'indirizzamento casuale estendibile è stato concepito per aggirare la necessità di precisare in anticipo le dimensioni della tabella d'indirizzamento casuale. Viene calcolato un codice d'indirizzamento più lungo del necessario e di esso si usa solo la parte occorrente per sistemare il numero di voci presenti al momento; il resto del codice costituisce una riserva nel caso in cui le dimensioni dell'archivio aumentino. Una descrizione minuta dell'indirizzamento casuale estendibile esula dall'ambito di questo articolo; diciamo solo che questo metodo consente di mantenere la rapidità dell'indirizzamento casuale, pagando solo un prezzo modesto in termini di spazio di memoria supplementare.

In tutte le varianti dell'indirizzamento casuale, le voci dell'archivio sono registrate nell'ordine arbitrario, determinato dalla funzione di miscuglio. Nel procedimento di memorizzazione va perduta qualsiasi relazione di prossimità sequenziale esistente fra gli elementi nell'insieme di dati iniziale. Per esempio se un dizionario viene registrato in una tabella d'indirizzamento casuale, le parole che cominciano per *c* vengono sparpagliate a caso per tutta la matrice e, di conseguenza, quando una parola viene recuperata non è possibile ottenere una risposta rapida a domande sui lemmi che nell'ordine alfabetico, erano adiacenti a quella parola.

a

ARCHIVIO NON ORDINATO

```
seltzer, excelsior beverage co. newark 242-0412
new york air, nyair 800-221-9300
usair 622-3201
seltzer, elliot beverage somerville 356-0273
united airlines (ua) 624-1500
imagen systems (laser printers) 496-7200
```

b

COMANDO: *grep* seltzer

USCITA:

```
seltzer, excelsior beverage co. newark 242-0412
seltzer, elliot beverage somerville 356-0273
```

c

COMANDO: *grep beverage < telnos*

USCITA:

```
seltzer, excelsior beverage co. newark 242-0412
seltzer, elliot beverage somerville 356-0273
```

d

COMANDO: *grep air < telnos*

USCITA:

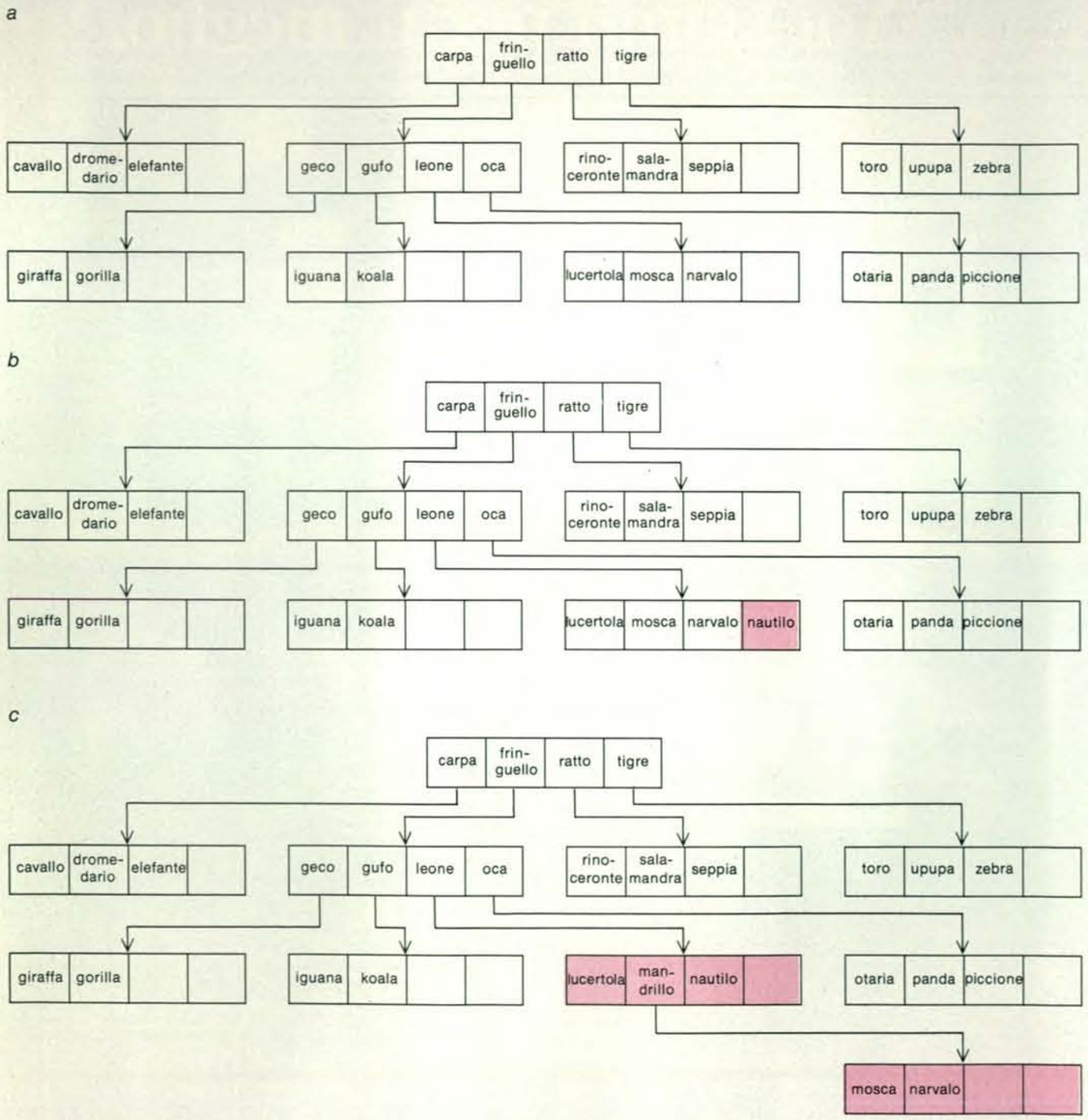
```
new york air, nyair 800-221-9300
usair 622-3201
united air lines (ua) 624-1500
```

Un archivio non ordinato è costituito da voci memorizzate senza alcun ordine particolare. Il riquadro superiore illustra un archivio di questo genere: un piccolo elenco telefonico chiamato *telnos (a)*. Quando viene dato il comando *grep* del sistema operativo Unix, tutte le righe dell'archivio vengono esplorate una dopo l'altra e vengono individuate tutte quelle che contengono una determinata chiave, o combinazione di simboli. I nomi delle ditte dell'elenco che forniscono acqua di seltz possono essere reperiti mediante la chiave *seltzer*, che è stata inserita all'inizio di entrambe le voci *(b)*. Queste due voci possono essere reperite anche mediante la chiave *beverage*, che si trova nei nomi di entrambe le ditte *(c)*. I nomi delle compagnie aeree che si trovano nell'elenco possono essere trovati mediante la chiave *air (d)*.

Un albero *B* offre la possibilità di rispondere in modo efficiente a domande su elementi che stavano vicini nella successione originale. L'albero *B* è uno strumento per attuare la ricerca binaria in cui le ripetute suddivisioni dell'archivio, invece di essere calcolate dall'algoritmo, sono incorporate nella struttura dei dati. L'albero *B* ha la forma di un albero capovolto: vi sono molte categorie (le foglie) in fondo e una sola categoria (la radice) in cima. Ciascuna categoria, o nodo, consta di un insieme di chiavi per le voci.

Al livello più basso dell'albero ciascun nodo contiene un gruppo di voci, disposte in ordine e senza omissioni. Al livello immediatamente superiore ciascun nodo contiene una sola chiave; queste chiavi provengono ciascuna da un nodo appartenente a un sottoinsieme dei nodi del fondo. Questo procedimento di riduzione continua fino alla cima dell'albero, dove si trova un unico nodo. L'albero viene percorso dalla cima al fondo e le chiavi di ogni nodo servono da puntatori verso i nodi del livello inferiore.

Se un dizionario inglese fosse registrato sotto forma di albero *B*, il primo nodo potrebbe contenere le parole *chromophore, epicycle, impolite* e così via, che costituiscono una famiglia di lemmi che suddividono il dizionario. *Chromophore*, che è la prima chiave, potrà puntare verso un nodo del secondo livello che contiene le parole *alfalfa, apocryphal, available, binocular, bully* e *celery*. Si vede subito che il secondo elenco contiene lemmi che suddividono il dizionario dal principio fino alla parola *chromophore*. Ciascun elemento del secondo gruppo punta verso un elenco in cui la risoluzione è più fine. Al fondo dell'albero i nodi puntano verso tutte le parole del dizionario.



La struttura dell'albero *B* e le strategie per aggiungere elementi all'albero sono qui illustrate per un piccolo archivio che contiene nomi e descrizioni di animali memorizzati in ordine alfabetico. L'albero *B* è un tipo di registrazione ordinata di dati, che consiste in nodi, ovvero gruppetti di chiavi. Ciascun nodo comprende chiavi che dividono l'archivio, o una parte di esso, in frazioni *(a)*. Il nodo in cima contiene chiavi che fungono da punto di divisione per l'archivio complessivo. Ciascuna chiave del nodo in cima punta verso un nodo al livello inferiore. Le chiavi di ciascun nodo inferiore riempiono gli spazi tra una chiave e l'altra del nodo superiore. Per esempio lo spazio tra *fringuello* e *ratto* è riempito da *geco, gufo, leone* e *oca*; la chiave *fringuello* del nodo superiore punta verso il nodo inferiore che comprende quei nomi. Per trovare la voce *lucertola*, viene esplorato il nodo in cima e si scopre che *lucertola* sta tra *fringuello* e *ratto*; quando viene esplorato il secondo nodo, si trova che *lucertola* sta fra *leone* e *oca*. *Leone* punta verso il nodo che comincia con *lucertola*. La chiave *lucertola* punta verso la voce che descrive l'animale (non rappresentata qui). L'aggiunta di un elemento a un albero *B* può essere un lavoro semplice o complesso, secondo la posizione delle caselle vuote. Aggiungere all'albero *nautilo* è immediato: la voce è inserita nella casella vuota dopo *narvalo (b)*. Una volta aggiunto *nautilo*, però, l'aggiunta di *mandrillo* comporta che il nodo che comincia con *lucertola* venga suddiviso in un nodo superiore e uno inferiore *(c)*. In tal modo l'ordine dei puntatori viene mantenuto: *mosca* e *narvalo* riempiono lo spazio fra *mandrillo* e *nautilo*.

L'uso degli alberi *B* si è diffuso per svariati motivi. Come si è già osservato, un albero *B* offre la possibilità di rispondere a domande relative agli elementi adiacenti a un elemento che sia stato già rintracciato. In più la memorizzazione mediante un albero *B* è piuttosto veloce: una ricerca richiede grosso modo $\log_2 n$ operazioni e l'aggiunta o la soppressione di voci richiedono anch'esse circa $\log_2 n$ operazioni.

Una delle ragioni principali dell'impiego così diffuso degli alberi *B* è legata alla struttura fisica della memorizzazione su disco magnetico. Per amor di semplicità si suppone spesso che tutte le ricerche in un archivio richiedano più o meno lo stesso tempo. Tuttavia è solo nella memoria centrale che tutte le ricerche richiedono un tempo fisso e le basi di dati interessanti hanno dimensioni tali da non poter essere contenute in una memoria centrale. Le basi grandi vengono memorizzate su dischi, per i quali esistono due tipi di ricerca, con tempi di recupero molto diversi. Il tempo di accesso casuale è il tempo medio necessario per reperire un record situato in una posizione arbitraria del disco. Il tempo di accesso sequenziale è il tempo occorrente per rintracciare il record che segue quello reperito per ultimo. In una macchina generica il tempo di accesso casuale può essere trenta volte maggiore del tempo di accesso sequenziale. Pertanto un programma efficiente rende massimo il numero delle ricerche sequenziali e minimo il numero di quelle casuali col recuperare gruppi di dati relativamente grandi ogni volta che venga fatta una ricerca. Se i dati sono memorizzati in un albero *B*, la dimensione dei nodi alla base dell'albero può essere adattata alla grandezza di un settore del disco. L'indirizzamento casuale, invece, non riesce a sfruttare bene la struttura del disco.

Nella discussione precedente, i metodi per memorizzare e per estrarre i dati sono stati considerati separatamente. Tuttavia, quando si progetta un sistema reale per conseguire i risultati operativi migliori è spesso necessario combinare parecchie tecniche, come confermano numerosi esempi da me incontrati tratti da lavori svolti in collaborazione con i miei colleghi. Un programma sperimentale da noi ideato fornisce le previsioni del tempo per qualunque città degli Stati Uniti. Se l'utente chiede che tempo farà in una data città, il programma trova la località più vicina a quella città in cui le condizioni meteorologiche siano state registrate e riferisce le osservazioni più recenti. Successivamente individua le previsioni meteorologiche più vicine e riferisce anche queste.

Questo servizio di informazioni meteorologiche comporta l'uso di tre basi di dati con i relativi programmi. Il sistema si appoggia principalmente a un circuito di comunicazioni del Servizio meteorologico nazionale americano, che fornisce ogni giorno sei megabyte di informazioni atmosferiche, comprese le osservazioni e le previsioni. Le osservazioni sono effettuate presso gli aeroporti, identificati solo da un codice di tre lettere; di conseguenza,

```
COMANDO:  $ date
          $ data

USCITA:   Fr. May 4 12:55:48 EDT 1984
          Ven. 4 maggio 1984 ore 12:55:48

COMANDO:  $ weather
          $ tempo
          elmira, ny
          elmira, ny

USCITA:   Elmira, NY (42.093 N, 76.807W)
          6,3 miles NW, at the airport in Elmira, NY (CHEMUNG COUNTY) (11:55 AM EDT):
          temperature 55°F, humidity 96, weather overcast, visibility 15 miles

          Elmira, NY (42,093 N; 76,807W)
          6,3 miglia a NW, aeroporto di Elmira, NY (CONTEA DI CHEMUNG) (ore 11:55):
          temperatura 55°F, umidità 96, cielo coperto, visibilità 15 miglia

          Next 48 hours at Rochester, NY (ROCHESTER-MONROE COUNTY)
          To 8 PM EDT/5 high 62 low 41, prob. precip. to 8 AM 30% to 8 PM 10%
          To 8 PM EDT/6 high 66 low 45, prob. precip. to 8 AM 40% to 8 PM 60%
          Per le prossime 48 ore a Rochester, NY (CONTEA DI ROCHESTER-MONROE)
          Fino alle 20 del 5: temp. max 62 °F min 41 °F, prob. di precipit. 30% fino
            alle 8, 10% fino alle 20
          Fino alle 20 del 6: temp. max 66 °F, min. 45 °F, prob. di precipit. 40% fino
            alle 8, 60% fino alle 20

          Forecast For Western New York
          National weather service buffalo ny
          430 am edt fr. may 4 1984
          Rain.. heavy in spots.. Becoming intermittent during the day from
          west to east and ending tonight. Highs in the mid to upper 50's
          today and lows tonight about 40. A mix of clouds and sunshine
          Saturday. Highs 60 to 65.
          Previsioni per lo Stato di New York occidentale
          Servizio meteorologico nazionale buffalo ny
          ore 4:30 ven 4 maggio 1984
          Pioggia.. Localmente forte.. Con tendenza alla variabilità.
          da ovest verso est durante la giornata e miglioramento
          in serata. Massima diurna fra i 55 e i 59 °F, minima notturna sui 40 °F.
          Sabato variabile con annuvolamenti sparsi. Massima dai 60 ai 65 °F.
```

Le informazioni meteorologiche per la città di Elmira (New York) sono qui presentate da un programma ideato dall'autore. Dopo aver controllato la data, l'utente chiede informazioni sul tempo in quella città. Il programma individua allora il punto più prossimo in cui sono state compiute le osservazioni (l'aeroporto di Elmira) e riferisce gli ultimi dati. Poi individua le previsioni meteorologiche più vicine: le previsioni per Rochester sono state fatte meccanicamente, quelle per Buffalo dall'uomo. Il programma per le informazioni meteorologiche, che è in grado di descrivere le condizioni atmosferiche per ogni città degli Stati Uniti, si basa soprattutto sulle osservazioni del Servizio meteorologico nazionale, compiute negli aeroporti. Di conseguenza, oltre alla base di dati che registra le informazioni meteorologiche stesse, ne occorrono altre due: una tabella che fornisca latitudine e longitudine di ciascun aeroporto e una che riporti latitudine e longitudine di ciascuna città. Quando viene fatta una richiesta, il programma individua latitudine e longitudine della città, trova gli aeroporti più prossimi e riferisce le previsioni e le osservazioni comunicate da questi. In colore viene data la traduzione italiana dell'originale.

oltre alla base di dati che contiene le informazioni meteorologiche, sono necessarie altre due basi di dati. La prima base ulteriore è una tabella che fornisce la latitudine e la longitudine di ciascun aeroporto ed è compilata in base ai registri della Federal Aviation Administration. La seconda è una tabella che fornisce la latitudine e la longitudine di ogni città degli Stati Uniti, ed è ricavata dai registri del Census Bureau.

I dati sulle città sono memorizzati sotto forma di albero *B*, mentre i resoconti meteorologici sono memorizzati in un archivio a secchielli, nel quale gli Stati Uniti sono suddivisi in quadratini secondo la latitudine e la longitudine. Quando arriva un gruppo di rilievi meteorologici, il codice dell'aeroporto viene trasformato in una località mediante l'albero *B* e le informazioni vengono registrate nel secchiello giusto. Quando viene richiesta un'informazione sul tempo, si cerca nel secchiello l'aeroporto più vicino. Se i dati fossero in forma unidimensionale e non bidimensionale, tutto il lavoro potrebbe essere svolto mediante gli alberi *B*, ma gli alberi *B* non sono molto adatti per accogliere dati bidimensionali. Al servizio meteorologico ricorrono sovente i miei collaboratori quando devono compiere un viaggio e vogliono sapere che tempo fa nel luogo di destinazione.

Un altro servizio che forniamo è uno strumento per scegliere le informazioni contenute nei servizi di attualità raccolti e archiviati dalla Associated Press. Il programma è ancora allo stadio sperimentale e al momento vi partecipa un centinaio di persone. Ogni giorno vengono memorizzate nel sistema informatico circa 200 000 parole. Esistono due metodi principali per accedere alle informazioni contenute nei servizi di attualità. Il primo metodo consente all'utente di scegliere le notizie di attualità da un menù visualizzato su schermo; le notizie vengono scelte in base ad alcune parole che servono da titolo e che compaiono sullo schermo video di un terminale. In media ogni giorno una quarantina di persone legge in complesso 840 servizi di attualità, scegliendoli dal menù.

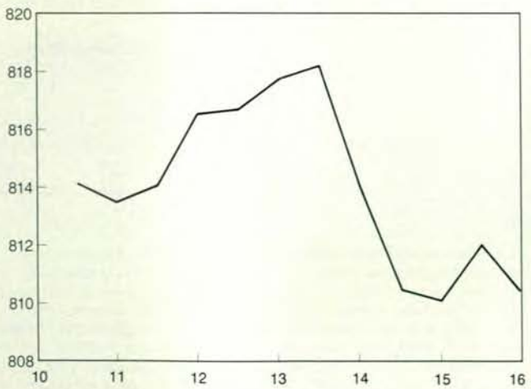
Il secondo metodo di accesso consente ai lettori di rintracciare i servizi con l'ausilio di un profilo costituito da parole, frasi o operazioni sintattiche particolari. Chi voglia leggere materiale sul Monte Everest può richiedere tutti i servizi dell'Associated Press in cui compaia la parola «Everest». Si può dare risposta anche a quesiti imperniati su locuzioni come «navetta spaziale» o su condizioni tipo la presenza nella stessa frase di «telefono» e «regolamentazione». Una cinquantina di persone hanno fatto richieste continuative di questo tipo e ogni giorno vengono distribuiti circa 550 servizi che si attagliano a un profilo particolare.

Tutti i sistemi che ho descritto finora sono concepiti per memorizzare e recuperare informazioni poste in forma di numeri o parole, ma è anche possibile elaborare grandi quantità di informazioni ricavate da immagini e grafici. Si consideri per esempio il problema di memorizzare la pianta di una città in un archivio informatico che debba poi servire per rispondere a quesiti a cui di solito si risponderebbe consultando la pianta. Le informazioni contenute in una pianta stradale possono essere ricondotte a due tipi principali di dati: l'ubicazione dei nodi e l'ubicazione dei rami. Un nodo è il punto in cui due vie si incrociano; un ramo è un segmento di via che collega due nodi. Le posizioni dei nodi e dei rami si potrebbero tradurre in forma digitale tramite dispositivi chiamati tavolette o analizzatori per la conversione digitale, ma fortunatamente questa trasformazione è già stata effettuata dal Census Bureau. Il nostro sistema per elaborare le piante stradali è quindi basato sui dati fornitici da quell'ufficio.

Come si fa a memorizzare la pianta? I dati sono numerosissimi e, ciò che è ancora peggio sotto il profilo della memorizzazione, sono bidimensionali. Il sistema di registrazione dev'essere in grado di conformarsi a queste due caratteristiche. Vi sono vari metodi di memorizzazione adatte e, per sceglierne uno, abbiamo considerato i quesiti ai quali il sistema avrebbe dovuto rispondere in modo efficiente. Sono quattro i tipi importanti di quesiti: trovare l'ubicazione di un edificio quando se ne conoscano la via, il numero e il codice postale; scoprire se due vie s'intersecano e, in caso affermativo, individuarne gli incroci; trovare tutti i punti che possono essere raggiunti direttamente da un punto dato; trovare tutte le vie situate a meno di una certa distanza da un punto dato.

Due strumenti per memorizzare i dati contenuti in un disegno bidimensionale sono: la matrice di connessione e l'albero *k-d*. Nella matrice di connessione l'unica informazione memorizzata è l'elenco di tutte le coppie di nodi che sono collegati da un ramo. Questo archivio non contiene informazioni sufficienti a far funzionare il sistema della pianta stradale, poiché un'informazione essenziale è costituita dai nomi delle vie e dall'ubicazione dei nodi. I nomi delle vie devono stare nel sistema affinché il programma possa stabilire quando l'itinerario passa da una via all'altra; l'ubicazione dei nodi deve starci per poter distinguere una svolta a destra da una svolta a sinistra e anche per poter costruire una rappresentazione materiale della pianta.

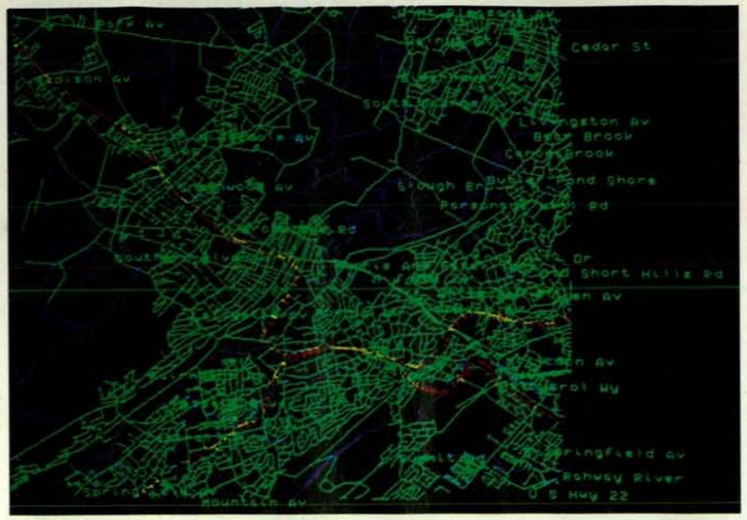
ORA	INDICE INDUSTRIALE DOW JONES
10:30	814,12
11:00	813,55
11:30	814,12
12:00	816,59
12:30	816,69
13:00	817,73
13:30	818,21
14:00	814,12
14:30	810,50
15:00	810,12
15:30	812,02
CHIUSURA	810,41

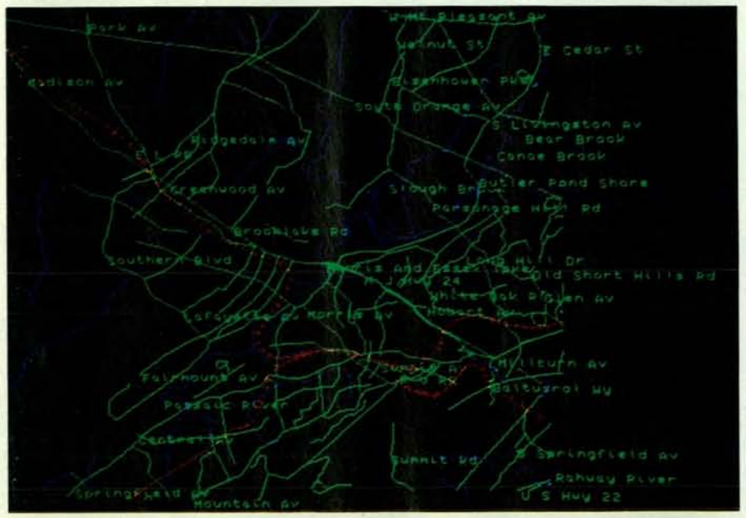


CALCOLATORE	«Il mercato ha avuto una tendenza al rialzo nella prima parte della giornata di ieri, ma è ricaduto poco prima della chiusura. Sulle quotazioni delle azioni vi sono stati contrasti e la Borsa ha subito una piccola perdita con uno scambio moderato.»

WALL STREET JOURNAL	«La Borsa ha chiuso con risultati contrastanti dopo che il tentativo di spingere il suo contraccolpo in una quarta sessione è divenuto esitante in uno scambio attivo e continuo.»

Il software per la borsa valori può così fornire i risultati degli scambi della giornata sotto forma di numeri, immagini visive o parole. L'indice industriale Dow Jones è disponibile in una forma leggibile dalle macchine; il riquadro in alto a sinistra mostra l'indice del 23 giugno 1982 a intervalli di mezz'ora. Il riquadro in alto a destra mostra l'uscita di un semplice programma che trasforma le informazioni numeriche in un grafico giornaliero dell'indice. Un programma più complesso, ideato da Karen Kukich della Carnegie-Mellon University, fornisce un riassunto in lingua inglese dell'andamento dell'indice durante la giornata. Nel riquadro in basso compare un confronto fra un riassunto prodotto dal calcolatore e il sommario degli scambi della stessa giornata pubblicato dal «Wall Street Journal». (I due testi sono tradotti in italiano.) Il programma per la borsa non contiene informazioni sugli scambi dei giorni precedenti e il riassunto non presenta generalizzazioni sulle tendenze che valgano per più di un giorno. Il generatore del testo è specializzato per il compito che svolge: per esempio non si esprime mai al futuro.





L'albero *k-d* è una variante dell'albero *B* che può accogliere anche dati a più dimensioni. Nell'albero *k-d* scendere da un livello al livello immediatamente inferiore corrisponde non solo a una selezione sempre più fine dei dati, ma anche a un cambiamento di dimensione. A ogni passo i dati vengono infatti divisi lungo la dimensione più grande. Così, per esempio, quando si memorizza una carta digitalizzata del Cile la prima suddivisione avviene lungo l'asse nord-sud, mentre per una carta del Tennessee la prima suddivisione viene compiuta lungo l'asse est-ovest.

L'albero *k-d* è uno strumento efficace per memorizzare dati bidimensionali, perché dividendo le informazioni lungo la dimensione maggiore viene ridotto il numero di decisioni che si debbono prendere per rintracciare un elemento. Tuttavia, il problema più serio nella memorizzazione di una pianta, non è quello di ridurre il numero delle decisioni per ogni operazione di reperimento, bensì quello di render minimo il numero dei settori del disco che debbono essere esaminati. Al pari della matrice di connessione, neppure l'albero *k-d* è in grado di memorizzare i nomi delle vie insieme con le vie e quindi i nomi debbono essere forniti da un archivio separato.

Per accelerare le operazioni di reperimento sul disco, è stato allestito un «archivio appezzato» che comprende due sottoarchivi contenenti informazioni di tipo diverso. Uno degli archivi è l'elenco principale dei rami, ricavato dai dati del Census Bureau e da una tabella di informazioni supplementari per ciascuna via. I dati del Census Bureau forniscono l'ubicazione e il nome di ciascun ramo; la tabella supplementare indica le vie a senso unico e le autostra-

Una base di dati geografica memorizzata in forma digitale genera piante a vari livelli di dettaglio. Ciascun riquadro illustra la stessa area: un quadrato di quattro miglia di lato centrato su un incrocio a Chatham, nel New Jersey (Stati Uniti). La pianta in alto riporta tutte le vie; quella di mezzo è stata sfoltita e riporta solo le strade più importanti; la pianta in basso è stata sfoltita ulteriormente e semplificata adottando l'ipotesi che ciascuna via proceda in linea retta fra gli incroci rimasti. Lo sfoltimento può ridurre di parecchio il tempo macchina occorrente per elaborare le piante: la pianta completa richiede 56 secondi di elaborazione su un VAX 11/750 della Digital Equipment Corporation (che è un potente minicalcolatore); la mappa sfoltita richiede 34 secondi e la mappa sfoltita e semplificata 5 secondi. I programmi per l'elaborazione delle mappe si basano su un archivio che comprende due sottoarchivi. Nell'archivio principale vi sono, in forma digitale, i dati del Census Bureau, relativi alla posizione di nodi e rami. Un nodo è un'intersezione e un ramo è un segmento di via che collega due nodi. Il secondo archivio è un «archivio appezzato» in cui la mappa è divisa in piccoli quadrati e, in ciascun quadrato, è registrata la configurazione dei nodi e dei rami. I due sottoarchivi sono utilizzati insieme per sfoltire le mappe e per rispondere a quesiti sui possibili percorsi.

INFORMATICA

LINGUAGGI:

A. Celentano
FORTRAN 77

G. Cioni, S. Crespi Reghizzi, M. Moscarini
PASCAL *(nuova edizione)*

I. Quartiroli, M. Fusaro, S. Smareglia
UNIX *(nuova edizione)*

B. Petrillo
BASIC. Programmazione strutturata e linguaggio

T. Plum
PROGRAMMARE IN C *(in preparazione)*

G. Gini, M. Gini, G. Guida
LISP

M. Tausel
FORTH *(in preparazione)*

M. Thorin
ADA

SCIENZA DEI SISTEMI:

P. Della Vigna, C. Ghezzi, R. Morpurgo
FONDAMENTI DI INFORMATICA

C. Baldissera, S. Ceri, A. Colorni
METODI DI OTTIMIZZAZIONE E PROGRAMMI DI CALCOLO

S. Crespi Reghizzi
SINTASSI, SEMANTICA E TECNICHE DI COMPILAZIONE

F. Tisato, R. Zicari
SISTEMI OPERATIVI
(nuova edizione in preparazione)

SCIENZA TECNICA E SOCIETÀ:

J.R. Searle (a cura di G. Tonfoni)
MENTI, CERVELLI, PROGRAMMI

G. Brand, B. Kündig e altri
IL CALCOLATORE NELLA ORGANIZZAZIONE DEL LAVORO
(due volumi)

M. Gambaro
L'INFORMAZIONE TELEMATICA

clup edizioni

de ad accesso limitato e fornisce informazioni sui limiti di velocità e sulla velocità media dei vari spostamenti.

Nell'archivio principale dei segmenti, come nei dati originali del Census Bureau, ciascuna via è divisa in segmenti piuttosto brevi, in modo che un segmento ne intersechi altri solo agli estremi e che ogni segmento possa essere approssimato da un tratto di linea retta. Un segmento corrisponde a un singolo record nella base di dati. Questa forma di organizzazione comporta che, per la maggior parte delle vie, ogni isolato richieda un record distinto. I record sono disposti in ordine alfabetico secondo il nome della via.

Ciascun record contiene un campo che indica se il segmento è una strada normale, un'autostrada ad accesso limitato, una rampa d'accesso oppure un altro elemento topografico, per esempio una linea ferroviaria, un fiume o una linea di confine. Il record di una via comprende anche i numeri civici su entrambi i lati del segmento, informazioni sulla velocità degli spostamenti, l'ubicazione degli estremi e il codice postale su entrambi i lati del segmento. Mediante una ricerca binaria nell'archivio principale è facile rintracciare l'ubicazione di qualunque strada ed elencare le intersezioni di due strade qualunque. Perciò già l'archivio principale da solo può rispondere ai primi due tipi di quesiti.

Il secondo archivio è un gruppo di segmenti organizzati secondo una struttura appezzata, in cui l'area rappresentata dalla pianta è divisa in appezzamenti quadrati di 10 000 piedi (circa 3000 metri) di lato. Per tracciare una pianta che comprenda tutti i segmenti di una data area, il programma prima esplora l'elenco degli appezzamenti nell'ordine e poi esplora nello stesso modo gli appezzamenti. Un segmento che compaia in un solo appezzamento viene registrato una sola volta, mentre un segmento che compaia in più appezzamenti viene registrato in tutti gli appezzamenti nei quali abbia un estremo. L'archivio appezzato è costruito completamente a partire dall'elenco principale. Quando vengono fatte delle variazioni, esse vengono introdotte soltanto nell'elenco principale e l'archivio appezzato viene poi rigenerato automaticamente.

Una volta scelte le tecniche per strutturare e memorizzare le informazioni della pianta si è dovuto risolvere il problema di introdurre un'interfaccia tra utente e sistema. La maggior parte delle interfacce per calcolatore esistenti sono meno buone di una carta stradale a stampa. Per molte carte stradali a stampa, infatti, il rapporto tra la larghezza del carattere tipografico più piccolo e la larghezza della carta è circa di 1 a 10 000, mentre per un terminale di calcolatore di qualità anche molto elevata il rapporto tra la larghezza della lettera più piccola e la larghezza dello schermo è solo di 1 a 125 circa. Inoltre la maggior parte dei dispositivi di elaborazione sono in grado di scrivere solo in senso orizzontale; alcuni possono scrivere anche in senso verticale, ma quasi nessuno è in grado di scrivere secondo angoli intermedi. A causa di queste limitazioni, molti nomi di strade devono essere omessi dalle piante che vengono generate a partire dalle informazioni digitalizzate.

Per stabilire quali indicazioni debbano essere omesse, il programma sfrutta informazioni sulle loro dimensioni e anche sull'importanza delle vie. Si fa l'ipotesi che quanto più lunga è una via tanto più importante essa sia come parte di un possibile itinerario; in pratica questa ipotesi si rivela valida. Le informazioni sull'importanza relativa delle vie vengono impiegate anche per compendiare o sfoltire le carte, in modo da rappresentare vaste aree senza troppi particolari e da tracciare itinerari che sfruttino solo le strade più importanti. Per un'elaborazione rapida, conviene supporre che ciascuna via segua un percorso rettilineo fra le informazioni che rimangono dopo aver effettuato lo sfoltimento.

Il sistema che abbiamo costruito può rispondere efficacemente ai quattro tipi di quesiti elencati sopra. Quando la base di dati viene combinata con un programma scritto da Jane Elliott dei Laboratori AT&T Bell, è possibile trovare l'itinerario più breve in termini di tempo oppure di distanza tra due punti della carta. La gran parte delle elaborazioni sulle piante fa uso della struttura ad archivio appezzato, che è l'equivalente dell'archivio a secchielli usato per i dati unidimensionali. L'archivio appezzato è un po' più lento dell'albero *B*, ma sfrutta meglio di quest'ultimo la struttura del disco. In generale la lista degli appezzamenti di una carta sta tutta in un solo segmento del disco e quindi il reperimento può essere compiuto molto rapidamente. Inoltre, la struttura ad appezzamenti è facile da capire, usare e aggiornare.

Per elaborare i dati bidimensionali esistono poche procedure tipo, tuttavia combinando varie tecniche il problema delle piante può essere risolto. Si potrebbero citare molti altri esempi interessanti di gestione di basi di dati, poiché gli strumenti che generano informazioni in forma leggibile dalle macchine si sono moltiplicati. Questo aumento, tuttavia, non è stato accompagnato da uno sviluppo adeguato dei programmi che consentono l'accesso alle informazioni. Non c'è dubbio che nei prossimi anni verranno ideati programmi più efficaci e più originali per la gestione delle informazioni. Benché gli esiti di questo sviluppo non si possano prevedere, è probabile che il suo corso sia guidato dai principi esposti nell'introduzione di questo articolo: la necessità di attagliare ciascun programma al contenuto delle informazioni e al modo in cui queste saranno usate e la necessità di sfruttare compiutamente la struttura dei dispositivi fisici in cui operano i vari programmi.

Software per il controllo di processo

Ha la funzione primaria di comunicare con dispositivi fisici e di governarli e deve mettere il calcolatore in grado di operare nei tempi dettati dagli avvenimenti del mondo reale

di Alfred Z. Spector

I sistemi di calcolatori che controllano i processi del mondo reale stanno rapidamente crescendo di numero. Essi trattano con cose come il controllo del traffico aereo, il sistema della segnalazione di percorsi equivalenti per le ferrovie, la distribuzione di energia elettrica, la rete telefonica, il pilota automatico e altri sistemi di controllo di un aereo, i movimenti di ascensori, il controllo di robot e di macchine utensili, l'ambiente interno di edifici, le linee di produzione negli impianti industriali, il volo di veicoli spaziali e così via. Una caratteristica distintiva di un calcolatore per controllo di processo è che la sua funzione primaria è quella di comunicare con il mondo fisico anziché con un operatore umano (per quanto esso poi mostri le informazioni sullo stato del processo a un operatore). Un'altra caratteristica è che un calcolatore per il controllo di processo non può imporre i propri ritmi, ma deve rispondere immediatamente agli eventi del mondo esterno in generale.

Un tipico sistema potrebbe essere dedicato al controllo di una colonna di frazionamento che separa componenti chimici leggeri da quelli pesanti, come in una raffineria di petrolio. In questa applicazione il calcolatore, diretto dal software, riceve informazioni sul livello e sulla velocità di flusso dei vari fluidi oltre che sulla temperatura e sulla pressione nella colonna; emette comandi per controllare questi fattori e di conseguenza per determinare la quantità e la qualità dei prodotti. Il sistema di controllo può anche essere programmato per ridurre al minimo il consumo di energia nell'impianto.

Qualunque sia l'applicazione, i collegamenti tra il calcolatore e il processo sono realizzati da sensori e da attuatori. Tipicamente un sensore registra dati analogici (per esempio variazioni di temperatura), che devono essere trasformati in dati digitali prima di essere presentati al calcolatore. Con alcuni tipi di sensori, è il software che a intervalli regolari richiede informazioni; con altri tipi è il sensore che interrompe il software a intervalli irregolari per presentare informazioni. Con tutta probabilità un programma per il controllo di processo comprende anche un dispositivo temporizzatore - un orologio - che può essere considerato un sensore. Un attuatore manipola i processi nel mondo reale o elettricamente o elettromeccanicamente. Nel controllare la temperatura un attuatore potrebbe dover alzare o abbassare l'interruttore di un ventilatore.

I collegamenti tra il calcolatore e gli operatori umani sono dispositivi di ingresso *(input)* e di uscita *(output)*. Una tastiera è il dispositivo di ingresso tipico. I moderni sistemi di calcolatori spesso dispongono anche di altri dispositivi di ingresso, come penne luminose oppure il *mouse*, per mezzo dei quali l'operatore può effettuare scelte puntando sullo schermo di visualizzazione. Lo schermo stesso è un dispositivo di uscita, che mostra informazioni, sotto forma di testo o sotto forma grafica, sullo stato del processo. Un'altra forma di uscita è una suoneria d'allarme che indica che qualche fase del processo ha bisogno di attenzione.

Il cuore di un calcolatore per il controllo di processo è un modello del processo reale. Il modello ha tre componenti, che chiamerò stato del modello, funzione di aggiornamento di stato e funzione di previsione. Lo stato del modello consiste nei dati che forniscono una descrizione completa del processo reale in ogni istante. La funzione di aggiornamento di stato trasforma uno stato del modello in un altro basandosi sulle informazioni fornite dai sensori. La funzione di previsione, se viene fornito uno stato del modello accurato, genera un insieme di comandi al calcolatore, mediante la cui esecuzione si ottiene qualche condizione desiderata nel processo controllato. Quello che è stato descritto in termini formali è un sistema di controllo a retroazione: il software riceve dati dai sensori, svolge le funzioni di aggiornamento di stato e di previsione e invia comandi agli attuatori. I risultati di quei comandi influenzano ulteriormente i dati ricevuti dai sensori.

Separato dal modello, ma essenziale per il funzionamento del sistema, vi è un piano strategico. Specifica la successione degli stati attraverso i quali dovrebbe passare il processo controllato. Per esempio, in un sistema di controllo del traffico urbano il piano specifica lo stato dei semafori come funzione del tempo e del flusso del traffico. Il piano può essere fornito da operatori umani o può essere generato dal software a partire da un insieme di obiettivi più astratti stabiliti da chi ha progettato il sistema.

Un dispositivo abbastanza semplice per controllare l'erogazione di calore all'interno di un edificio illustra bene la struttura di un sistema di controllo di processo. L'hardware include un sensore per registrare la temperatura esterna, sensori installati in vari ambienti per registrare le temperature interne, un orologio e due attuatori, in qualità di interruttori per una pompa di calore e per una caldaia. Assumiamo che il software abbia due scopi: mantenere una temperatura in funzione

I sistemi di energia elettrica sono controllati dal software di un calcolatore, che visualizza lo stato del sistema in ogni dato istante. Nella fotografia in alto della pagina a fronte il software sta riportando le condizioni di un impianto di produzione di energia elettrica con due generatori di vapore e una turbina. La fotografia in basso mostra il sistema di trasporto dell'alta tensione della Iowa Electric Light and Power Company. Per mezzo di questa visualizzazione l'operatore del centro di controllo a Cedar Rapids può aprire e chiudere interruttori per dare potenza alle varie sottostazioni indicate nell'immagine. Il programma è stato installato dalla Aydin Controls.

dell'ora del giorno e minimizzare il consumo di energia.

Lo stato del modello include la temperatura interna ed esterna e l'ora del giorno. La parte più importante della funzione di aggiornamento di stato calcola una media ponderata dei dati provenienti dai vari sensori della temperatura interna. La funzione di previsione utilizza lo stato del modello, insieme con le informazioni intorno alle perdite di calore dell'edificio e al rendimento termico delle due apparecchiature di riscaldamento per prevedere quando ciascuna debba accendersi o spegnersi. La strategia richiede che si determini, per ogni periodo particolare, se convenga usare la caldaia o la pompa di calore, in base al relativo rapporto tra costo e benefici; per questo ci si potrebbe benissimo basare sulla temperatura esterna e sul costo del combustibile.

Si potrebbe estendere il sistema aggiungendo un maggior numero di sensori (per esempio per registrare i livelli del combustibile), tenendo conto di altri aspetti dello stato del modello e riorganizzandoli per dare messaggi di condizioni anomale (come il cattivo funzionamento di un'apparecchiatura di riscaldamento o l'apertura di una finestra). Questi perfezionamenti, comunque, non inciderebbero sulla natura fondamentale del sistema, che continuerebbe a basarsi su un modello del processo controllato e continuerebbe a impiegare una funzione di previsione per raggiungere nuovi stati.

La maggior parte dei sistemi di controllo di processo sono più complessi di quanto questo esempio possa suggerire. La ragione principale sta nella complessità del modello interno. Consideriamo un sistema di controllo per veicoli, molto elementare, che possa «sentire» solo le accelerazioni. Proprio per mantenere la velocità giusta la funzione di aggiornamento di stato deve operare un'integrazione matematica per ogni lettura dell'accelerazione. Se venisse aggiunta una telecamera per individuare ostacoli e seguire il percorso, l'analisi della visione per aggiornare lo stato del modello richiederebbe le più raffinate fra le tecniche dell'intelligenza artificiale.

Possono essere necessarie procedure complesse anche per costruire funzioni di previsione e per sviluppare piani strategici. La funzione di previsione che calcola gli angoli per un braccio a sei gradi di libertà di un robot per posizionarne e orientarne la «mano» richiede molta algebra lineare. Pianificare la successione degli stati intermedi necessari al braccio di un robot per muoversi con regolarità da una posizione all'altra è un'impresa ancora più difficile.

Calcoli e pianificazioni sono compiti che devono essere eseguiti in molte applicazioni del calcolatore e il software per il controllo di processo impiega tecniche per eseguirli che sono comuni ad altri tipi di software. D'altro canto, i sistemi di controllo di processo hanno requisiti che differiscono da quelli di altre applicazioni del calcolatore. Uno di questi requisiti riguarda la velocità. L'esatto svolgimento temporale delle attività di un calcolatore che gioca a scacchi o che calcola stipendi di rado è un fattore critico; potrebbero far comodo velocità più alte, ma ottenere il risultato qualche minuto (o qualche ora) prima o dopo non fa molta differenza. Invece un sistema che controlla un aviogetto deve prendere decisioni in tempi rapidi: deve agire in «tempo reale».

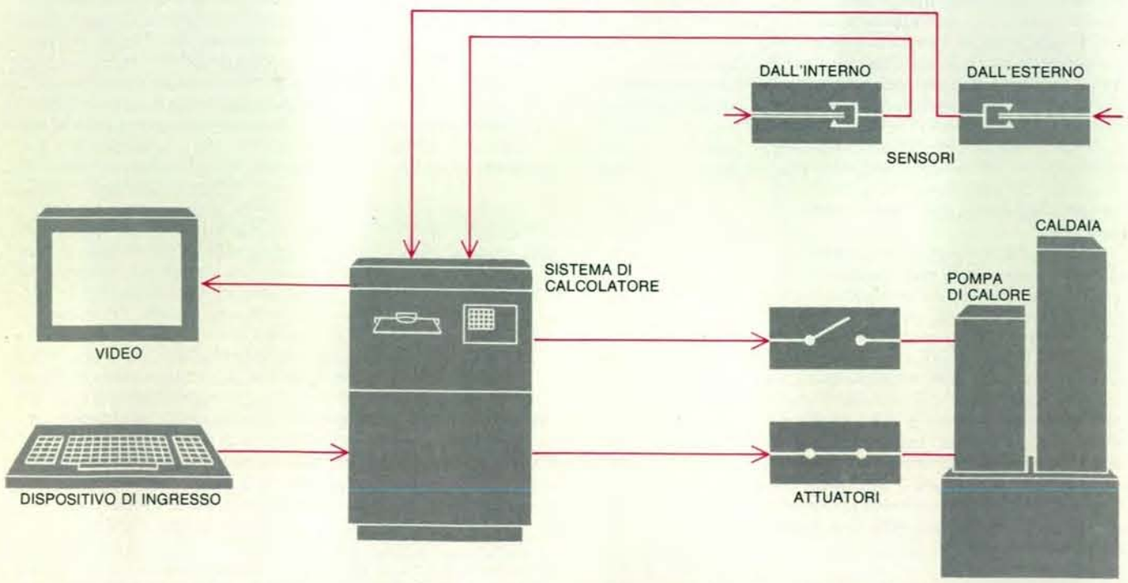
Analogamente, il calcolatore che gioca a scacchi o che prepara gli stipendi può svolgere un compito per volta e può distribuirne l'esecuzione nel tempo e nel modo più conveniente. Il sistema di controllo dell'aviogetto deve soddisfare più richieste così come esse si presentano, sincronizzando completamente il suo lavoro con i vari compiti. L'affidabilità è inoltre più importante nel caso dell'aereo, perché la conseguenza di un errore di programmazione potrebbe essere la perdita di vite umane e non puramente la sconfitta in una partita oppure una perdita finanziaria. In molti casi le esigenze di velocità, sincronizzazione e affidabilità sono rese più complesse dall'organizzazione fisica del sistema: i calcolatori, i sensori, gli attuatori possono essere separati spazialmente e possono trovarsi a operare in un ambiente difficile.

I problemi della sincronizzazione e della temporizzazione sono suggeriti dal semplice frammento di programma nell'illustrazione in alto della pagina a fronte. È parte di un programma che controlla il riscaldamento e ha la forma di un ciclo: una successione di istruzioni che viene eseguita ripetutamente. L'unica temporizzazione richiesta è che le procedure debbono essere eseguite abbastanza rapidamente per permettere una frequenza operativa specifica. Tuttavia se i sensori da interrogare o gli attuatori da comandare fossero molti il flusso di controllo nel software sarebbe molto più complesso. Inoltre, se il sistema dovesse gestire interruzioni asincrone da parte dei sensori e dovesse emettere comandi in risposta a tali eventi, il software non potrebbe più essere organizzato sotto forma di ciclo e neppure come un insieme di cicli, ma dovrebbe avere una topologia più complessa.

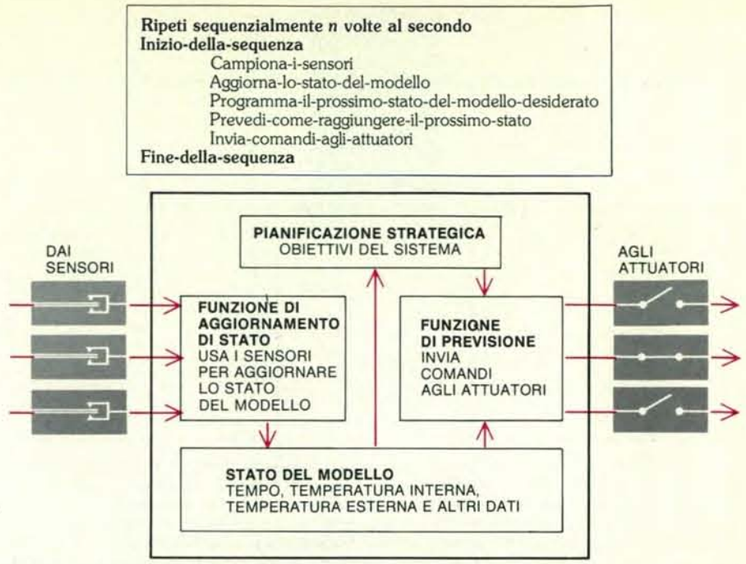
Il software per il controllo di processo è abitualmente costruito come una collezione di compiti cooperativi, ma indipendenti. Un compito o *task* è definito come una successione indipendente di istruzioni che possono richiamare dati almeno parzialmente separati dai dati collegati ad altri compiti. Si possono portare a compimento più compiti su più elaboratori, con ciascun elaboratore che esegue un singolo compito. Una disposizione più comune impiega un sistema operativo *multitasking* per organizzare nel tempo l'esecuzione di molti compiti su un singolo calcolatore. È necessaria un'analisi attenta per assicurare che i singoli compiti siano serviti in modo tale che siano soddisfatti gli obiettivi globali del sistema.

Una tecnica semplice organizza i compiti nel tempo a «giri di giostra»: a ciascun compito spetta un giro durante il quale viene eseguito fino al completamento. Con questa tecnica un compito può andare incontro a ritardi pesanti, se si trova accodato a compiti lunghi. Una seconda possibilità è una tecnica a giostra con prelazione, in cui a ogni compito è assegnato soltanto un breve periodo di esecuzione prima che l'unità di elaborazione si rivolga a un altro compito. Se rimane del lavoro da fare per il primo compito, gli viene assegnato un altro periodo di esecuzione al giro successivo. Un terzo metodo è quello della pianificazione basata sulla priorità, nella quale i compiti con priorità più alta ottengono periodi di esecuzione più lunghi, oppure più frequenti. Un'ultima impostazione importante è la pianificazione a scadenza *(deadline scheduling)*. Si stabilisce una scadenza ultima *(deadline)* entro la quale il compito deve essere completato e il sistema cerca di attendere ai vari compiti in modo che tutti raggiungano i loro obiettivi.

Un frammento di un semplice programma di controllo a più compiti è mostrato nell'illustrazione di questa pagina in basso. Sono coordinati due compiti: Leggi-tastiera-sensore, con priorità elevata, e Programma-prevedi-attua, con bassa priorità. Il compito ad alta priorità riceve segnali in ingresso in modo asincrono (cioè a intervalli non prevedibili) da un operatore umano e da sensori; l'altro compito regola periodicamente gli attuatori in risposta alle informazioni in ingresso recenti. Quando la tastiera o qualche sensore periferico segnala che



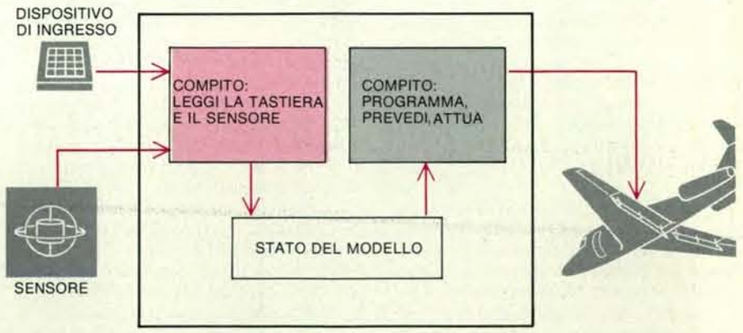
Il sistema di controllo di processo per regolare l'erogazione di calore a un edificio consiste in due unità di riscaldamento (una caldaia e una pompa di calore), di una schiera di sensori per registrare la temperatura esterna e la temperatura negli ambienti dell'edificio, di attuatori per accendere e spegnere le unità di riscaldamento, di un calcolatore e di un programma al quale sono stati imposti due obiettivi: mantenere una temperatura in funzione dell'ora del giorno e minimizzare il consumo di energia in ogni istante. Il programma opera secondo un modello tripartito della situazione reale. Lo stato del modello include le temperature interne ed esterne e l'ora; la funzione di aggiornamento di stato calcola una media ponderata delle varie temperature e revisiona lo stato del modello conseguentemente; la funzione di previsione tiene conto di problemi come lo stato del sistema e il tasso di perdita di calore dell'edificio per prevedere quando una delle unità di riscaldamento debba essere accesa o spenta. Il programma del calcolatore si basa sulla strategia di utilizzare solo l'unità più economica a meno che non siano necessarie entrambe.



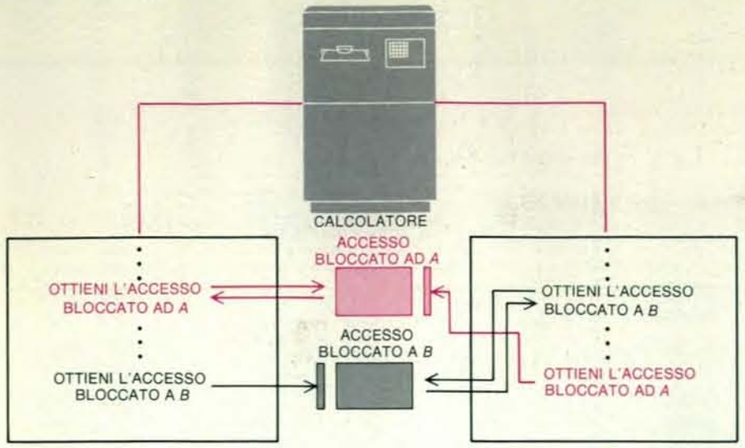
La struttura del software dello schema di controllo di processo per il sistema di riscaldamento è suggerita dal frammento di programma in alto. I comandi in neretto stanno per parole chiave del linguaggio di programmazione; gli altri comandi sono chiamate a procedure software che sono specifiche del sistema da controllare. I programmi hanno la forma di un ciclo: una successione di comandi che si può eseguire ripetutamente. In basso nell'illustrazione sono indicati i principali componenti del programma di calcolatore per controllare il sistema di riscaldamento.

Inizio-del-compito con priorità 10 {Chiamato Leggi-tastiera-sensore}
Attendi-comunicazione-da-tastiera
Leggi-tastiera
Leggi-sensore
Aggiorna-lo-stato-del-modello
Fine-del-compito

Inizio-del-compito con priorità 5 {Chiamato Programma-prevedi-attua}
Ripeti sequenzialmente 10 volte al secondo
Inizio-della-sequenza
Programma-il-prossimo-stato-del-modello-desiderato
Prevedi-come-raggiungere-il-prossimo-stato
Invia-comandi-agli-attuatori
Fine-della-sequenza
Fine-del-compito



Il sistema di controllo multicompito si basa su un singolo programma che segue e coordina parecchie funzioni; nell'esempio guida un aereo controllato a distanza. Un frammento del programma *(in alto)* include due compiti; uno con priorità elevata *(in colore)* e l'altro con priorità più bassa *(in grigio)*. Il compito a priorità elevata riceve l'ingresso a intervalli irregolari da un operatore che lavora su una tastiera e dai sensori sull'aereo. Il compito a priorità bassa regola gli attuatori che operano sui controlli dell'aereo. Il compito a priorità elevata non può essere interrotto. Se il compito a priorità bassa è in esecuzione quando arrivano ingressi dalla tastiera o da un sensore, esso viene interrotto dal programma. Entrambi i compiti hanno accesso allo stato del modello nella memoria del calcolatore. La condivisione di memoria è comune nella comunicazione tra compiti per i sistemi di controllo di processo diretti da un solo calcolatore.



Il problema del punto morto *(deadlock)* può sorgere in un programma che impiega la tecnica detta degli accessi bloccati *(locking)* per sincronizzare la condivisione di memoria da parte di più compiti. In questa tecnica un compito non può avere accesso ai dati in memoria fino a che non ottiene un blocco, il che significa che richiede al sistema operativo il diritto d'accesso e lo ottiene. Quando il compito non ha più bisogno dell'accesso, libera il blocco. Qui il punto morto è mostrato per un sistema a due compiti. Il compito a sinistra sta eseguendo una sequenza di istruzioni (rappresentata dai puntini) quando incontra il comando per ottenere l'accesso bloccato ad *A*. Lo ottiene e continua l'esecuzione. Nel frattempo il secondo compito ha seguito le istruzioni per ottenere l'accesso bloccato a *B*. Ora il primo compito emette un'istruzione per ottenere l'accesso bloccato a *B;* non può ottenerlo perché l'altro ha già questo blocco, così il primo compito rimane sospeso in attesa finché l'accesso bloccato non si è reso disponibile. Se il secondo compito ora invia una richiesta per ottenere l'accesso bloccato ad *A*, il sistema si arena in un punto morto.

sono disponibili nuovi dati, il sistema operativo, se è in esecuzione il compito a bassa priorità, lo interrompe e gli concede di riprendere solo dopo che sono stati letti i dati in ingresso. Quasi tutti i sistemi a controllo di processo hanno questa organizzazione multicompito.

È significativo in questa organizzazione che il compito Leggi-tastiera-sensore trasferisca i dati al compito Programma--prevedi-attua per mezzo dello stato del modello. Il primo compito carica semplicemente informazioni dalla tastiera nell'area di memoria dove è immagazzinato lo stato del modello, un'area alla quale ha accesso anche il secondo compito. La condivisione della memoria in questo modo è una tecnica efficace e potente per la comunicazione tra compiti. Questo è un procedimento tipico nelle applicazioni per il controllo di processo confinate su un solo calcolatore.

D'altro canto, la comunicazione tra compiti per mezzo della memoria condivisa complica l'elaborazione multicompito e porta ad alcuni problemi interessanti. Uno è la sincronizzazione. Supponiamo che, nel programma a due compiti, il compito a priorità bassa si interrompa dopo aver letto solo alcuni elementi dello stato del modello. Quando il compito riprende, legge i valori rimanenti e calcola un risultato determinato da tutte le letture. Durante l'interruzione, tuttavia, lo stato del modello può essere stato cambiato dall'altro compito, cosicché il calcolo si basa su una chimera di vecchi e nuovi dati. Supponiamo che il compito interrotto stesse misurando una posizione in un sistema di coordinate *x-y-z*, e che sia stato interrotto dopo aver letto *x*, ma prima di aver potuto leggere *y* e *z*; baserebbe i suoi calcoli su una posizione che non è mai esistita nel mondo reale.

Sono disponibili parecchie tecniche per sincronizzare l'accesso alla memoria condivisa, ma le più comunemente usate sono due, molto semplici. Una tecnica definisce segmenti di codice di programma nei quali un compito non può essere interrotto; le procedure di interruzione di un compito possono facilmente essere abilitate o disabilitate dal software del sistema operativo in sintonia con l'hardware del calcolatore. Tornando nuovamente all'esempio dei due compiti, il sistema può specificare che il compito Programma--prevedi-attua deve essere libero da interruzioni mentre sta leggendo le coordinate *x, y* e *z*. L'altra tecnica fondamentale si chiama bloccaggio *(locking)*. Prima che un compito possa ottenere l'accesso ai dati deve chiedere il diritto di farlo al sistema operativo. Quando il compito ha finito con i dati, dice al sistema operativo che i dati sono liberi per l'uso da parte di altri compiti. Si dice che il compito ha acquisito la capacità di blocco *(lock)* prima di poter leggere i dati nella memoria condivisa e che deve liberare il blocco dopo che ha finito. Con questo meccanismo il sistema operativo si assicura che solamente un compito alla volta abbia accesso ai dati.

Sebbene la sincronizzazione dell'accesso alla memoria condivisa sia concettualmente semplice essa è la principale sorgente di errori nei programmi multicompito. In parte il motivo sta nel fatto che gli errori di sincronizzazione sono estremamente difficili da identificare; possono non esibire sintomi eccetto che in rare circostanze. Il malfunzionamento del calcolatore che ritardò la prima missione della navetta spaziale consisteva in un problema di sincronizzazione (certamente molto complesso) che aveva la probabilità di sorgere una sola volta ogni 65 volte che il sistema veniva avviato.

I problemi connessi alla sincronizzazione della memoria condivisa possono inficiare l'affidabilità di un sistema in altri modi. Che cosa accade se un errore di programmazione porta un compito in un ciclo senza fine proprio mentre si trova in un periodo di non interruzione? A meno che non ci si sia presa molta cura nella progettazione del sistema, gli altri compiti potrebbero venire completamente esclusi dall'esecuzione. Un problema che può sorgere con i blocchi è che un insieme di compiti può formare una catena chiusa, in cui ciascun compito è in attesa che un altro liberi il blocco, col risultato che nessun compito viene eseguito. A un punto morto del genere può arrivare anche un sistema con due compiti e due blocchi solamente.

In un sistema a memoria condivisa sono anche possibili difficoltà non collegate alla sincronizzazione. Poiché la memoria è condivisa, i compiti non sono isolati l'uno dall'altro ed è difficile limitare gli effetti di una caduta del sistema. L'esecuzione difettosa può alterare lo stato di modello e di conseguenza disgregare l'operatività di molti altri compiti. In una rete di calcolatori geograficamente separati la condivisione della memoria diventa difficilmente gestibile per un ulteriore motivo, relativo all'hardware, anziché al software: semplicemente non è possibile costruire una memoria condivisa da parecchi calcolatori distanti tra loro.

Per queste ragioni, tra le altre, alcuni sistemi di controllo di processo sono organizzati come una collezione di compiti che non devono condividersi implicitamente lo stato di modello, ma invece si scambiano esplicitamente informazione trasmettendo messaggi. Ciascun compito conserva traccia di quegli elementi dello stato di modello di cui necessita nel proprio segmento non condiviso di memoria. Il sistema operativo fornisce gli strumenti per inviare e ricevere messaggi.

Per applicazioni semplici di controllo di processo l'organizzazione a trasferimento di messaggi è più complessa e abitualmente meno efficiente della memoria condivisa. Però il flusso esplicito d'informazione tra i compiti e il loro maggior isolamento offrono certi vantaggi. Un'organizzazione a trasferimento di messaggi è la sola scelta possibile quando un processo deve essere controllato con un sistema di parecchi calcolatori cooperanti senza memoria condivisa. Nonostante i pregi del trasferimento di messaggi, un programma può ancora giungere a un punto morto nel caso i compiti stiano tutti aspettando messaggi l'uno dall'altro.

I sistemi composti da parecchie macchine che lavorano di concerto stanno aumentando d'importanza e di numero, non solo nel controllo di processo, ma anche nei calcoli scientifici, nell'elaborazione di dati, nell'intelligenza artificiale. In alcune organizzazioni di questo genere numerosi elaboratori condividono una memoria comune; la condivisione è possibile, comunque, solo se i calcolatori sono fisicamente vicini. (Trasmettere un segnale a un chilometro di distanza implica un ritardo di parecchi microsecondi, che è del tutto intollerabile per l'alto volume di traffico tra l'unità di elaborazione centrale e la memoria centrale.)

Quando i calcolatori sono fisicamente separati, il sistema di controllo di processo è organizzato come una collezione di elaboratori, ciascuno con la propria memoria locale; gli elaboratori sono collegati in rete attraverso canali di comunicazione. Questi sistemi distribuiti, come vengono chiamati, si basano necessariamente sul trasferimento di messaggi. La distanza esistente tra gli elaboratori, i sensori e gli attuatori distingue i diversi tipi di sistemi distribuiti per l'effetto che essa ha sulla larghezza di banda e sulla latenza. La larghezza di banda è la misura del numero di bit che si può trasmettere in un secondo; la latenza è il ritardo tra l'invio e la ricezione d'informazione. Sulle brevi distanze si possono usare larghezze di banda maggiori e la latenza è bassa, permettendo così un'interazione più stretta tra i compiti che devono essere eseguiti dai calcolatori.

Un motivo frequente per l'allestimento di sistemi distribuiti nel controllo di processo è che i sensori e gli attuatori sono a loro volta distribuiti geograficamente. In tali situazioni è spesso possibile organizzare il sistema in modo tale che la maggior parte dell'elaborazione sia effettuata vicino ai sensori e agli attuatori più importanti e la comunicazione tra i calcolatori sia ridotta al minimo. Un esempio è un sistema di controllo di processo per una grande fabbrica con molti calcolatori semiautonomi in edifici diversi. I sistemi occasionalmente scambiano messaggi allo scopo di coordinare l'intero impianto e programmare i tempi di lavorazione, ma principalmente operano in modo indipendente. Il sistema in un edificio potrebbe controllare in maniera indipendente la produzione di un certo bene, ma comunicherebbe con altri sistemi per raccogliere informazioni sulle quote di produzione o sull'approvvigionamento di materie prime.

L'elaborazione distribuita può anche semplificare la progettazione e l'installazione del sistema (per esempio riducendo la quantità di cavi) e incoraggia una struttura organizzativa nella quale il personale responsabile di un processo abbia anche in carico i compiti di calcolo relativi. Le prestazioni potenzialmente più elevate costituiscono un'altra delle principali motivazioni per sistemi multicalcolatore. Da un punto di vista teorico si può eseguire una maggior quantità di calcoli se molti elaboratori lavorano contemporaneamente. Un sistema di controllo di processo organizzato come un insieme di compiti che comunicano per mezzo di messaggi è un'applicazione naturale di tale elaborazione parallela. Un compito di pianificazione che richieda grandi quantità di calcoli potrebbe venir eseguito su un elaboratore ad alta velocità che riceva dati da compiti che stanno girando su altri calcolatori e trasferisca loro i piani strategici elaborati.

Il motivo forse più importante per adottare l'elaborazione distribuita in un sistema di controllo di processo è che si tratti di un mezzo utile per ottenere affidabilità. Se il sistema è diviso in due sottosistemi che operano in modo autonomo, un guasto su una macchina non causerebbe la fermata dell'intero sistema. L'organizzazione della fabbrica che ho descritto sopra offre un esempio: persino se un sottosistema si guasta, gli altri sottosistemi possono continuare, almeno finché hanno a disposizione le materie prime.

Avendo come obiettivo il funzionamento continuo dell'intero sistema, un mezzo necessario, ma non sufficiente, per raggiungerlo è l'elaborazione basata sulla ridondanza. L'intero sistema a controllo di processo, dai sensori agli attuatori, e non solo il calcolatore e il suo software, deve continuare a lavorare nonostante un malfunzionamento. Un calcolatore che funzioni correttamente, ma che riceva dati non corretti dai sensori e che quindi emetta comandi errati agli attuatori, può fare più danni di un calcolatore che si fermi completamente.

La meta più alta nella ricerca dell'affidabilità è il funzionamento continuo completamente insensibile a malfunzionamenti locali. La natura dei compromessi a cui si deve scendere quando questa meta non può essere raggiunta è suggerita dal metodo seguito nei sistemi di controllo della navetta spaziale. A causa della ridondanza nel sistema di calcolo primario a bordo della navetta, un singolo guasto non costringe a variazioni nella missione. Un secondo guasto non arriva ancora a mettere a repentaglio la vita dell'equipaggio o l'integrità del veicolo, ma la navetta è riportata a terra il più presto possibile perché ulteriori guasti potrebbero essere rischiosi.

Una proprietà alla quale è stato attribuito il nome piuttosto sontuoso di «degradazione leggiadra» è quella per cui un

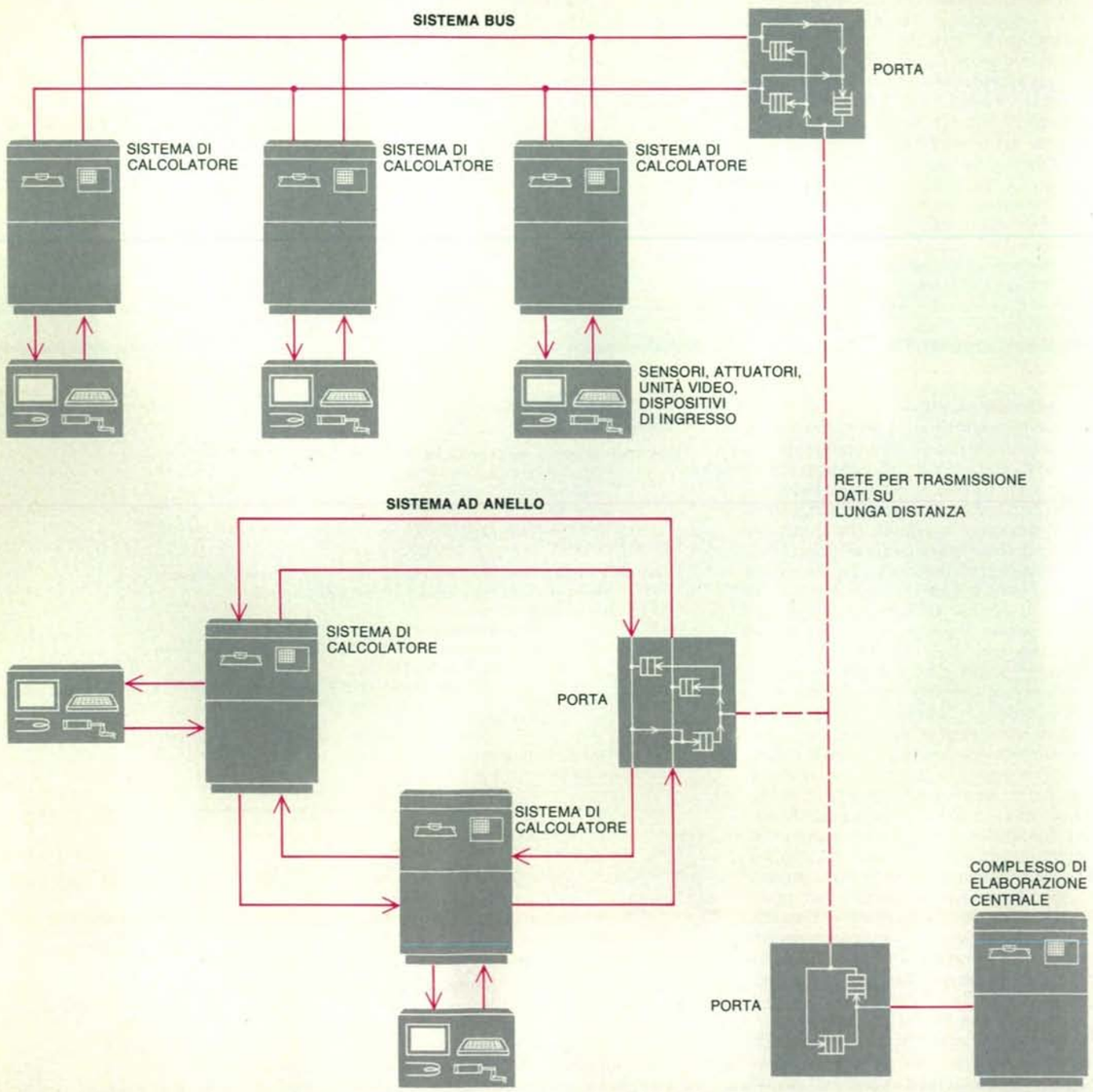
```
Inizio-del-compito con priorità 10 {Orologio}
        Ripeti sequenzialmente 10 volte al secondo
        Inizio-della-sequenza
                Invia-messaggio (Compito-programma-prevedi-attua, Tempo-trascorso)
        Fine-della-sequenza
Fine-del-compito

Inizio-del-compito con priorità 10 {Chiamato Leggi-tastiera-sensore}
        Attendi-comunicazione-da-tastiera
        Leggi-tastiera
        Leggi-sensore
        Invia-messaggio (Compito-programma-prevedi-attua, Nuovi-dati)
Fine-del-compito

Inizio-del-compito con priorità 5 {Chiamato Programma-prevedi-attua}
        Ripeti sequenzialmente per 10 volte al secondo
        Inizio-della-sequenza
                Attendi-messaggio
                Se il messaggio è dal Compito-temporizzatore allora
                Inizio-della-sequenza
                        Prevedi-come-raggiungere-lo-stato-successivo
                        Invia-comandi-agli-attuatori
                Fine-della-sequenza
                        Altrimenti Aggiorna-lo-stato-locale-con-nuovi-dati
        Fine-della-sequenza
Fine-del-compito
```

Il passaggio di messaggi è un'alternativa alla condivisione della memoria per sincronizzare il lavoro di più compiti in un sistema di controllo di processo. È il solo metodo adottabile quando il sistema ha parecchi calcolatori distanti. In alto appare una parte del programma che passa messaggi per controllare un aereo; l'organizzazione del programma è rappresentata in basso.

intero sistema se non può essere mantenuto in funzione dopo un guasto, può tuttavia possedere ancora una funzionalità parziale. Per esempio, un sistema che non può più controllare automaticamente un processo potrebbe ancora accettare comandi da un operatore via tastiera, in modo tale che il processo si possa controllare manualmente. Se non è possibile neppure un funzionamento parziale, il sistema di controllo dovrebbe essere in grado almeno di arrestare ordinatamente il processo, nell'eventualità di un guasto rilevante.

Vi sono almeno tre modi in cui possono verificarsi malfunzionamenti nel software. In primo luogo, i requisiti del sistema di controllo di processo possono essere stati stabiliti male, cosicché, anche se i programmi soddisfacessero perfettamente le specifiche, porterebbero a un funzionamento errato. Nella prima missione della navetta spaziale la conoscenza insufficiente delle caratteristiche di volo del veicolo portò a una traiettoria ascensionale che avrebbe potuto non permettere un atterraggio d'emergenza in Spagna, come parte del piano per le situazioni critiche. In secondo luogo, il progetto logico sottostante al software, o le istruzioni di programma in cui si realizza il progetto, possono non soddisfare le specifiche. L'imperfezione può essere anche molto piccola, per esempio un semplice errore tipografico. In terzo luogo, un operatore umano può commettere un errore nell'uso del software.

La protezione contro i guasti è un argomento chiave in tutte le applicazioni dei calcolatori, ma nel controllo di processo l'elevata probabilità di errori nella sincronizzazione e nella temporizzazione



Il sistema distribuito è una disposizione comune quando i sensori e gli attuatori di un sistema di controllo di processo sono diffusi su una vasta area. Il sistema mostrato abbraccia tre grappoli di calcolatori che sono geograficamente lontani tra loro. Un grappolo è composto da tre calcolatori che comunicano l'un l'altro per mezzo di una rete locale organizzata come una «pista» o un «bus» (un insieme di conduttori rettilinei e paralleli). Un altro grappolo impiega una rete ad anello. Il terzo componente del sistema è un calcolatore centrale preposto al coordinamento e alla raccolta di dati, che accetta comandi da operatori umani. I tre sistemi scambiano messaggi su una rete a grande distanza.

e i costi potenzialmente più elevati a essi associati rendono il compito di protezione dagli errori più importante e più difficile nello stesso tempo. Talvolta si possono applicare metodi analitici formali per dimostrare che un programma soddisfa i requisiti stabiliti. Nella maggior parte dei casi, comunque, servono metodi analitici meno formali. Un metodo adottato comunemente è quello di avere esperti nell'analisi dei requisiti ed esperti di programmazione che congiuntamente studiano le specifiche dei requisiti e i programmi preposti a soddisfarli.

Per quanto un programma sia stato analizzato in profondità, è tuttavia ancora necessario sottoporlo a verifica. I metodi analitici in generale non sono abbastanza potenti per catturare tutti gli errori concepibili nella progettazione del software e nella stesura dei programmi. Inoltre nessun confronto, per quanto esteso, tra i programmi e i loro requisiti può assicurare che i requisiti medesimi siano corretti e sufficienti.

Talvolta il personale che verifica i programmi è organizzato in un gruppo separato dai programmatori ed esegue controlli sulla funzionalità dei programmi in modo indipendente. Durante lo sviluppo del sistema operativo primario a bordo della navetta spaziale, il gruppo indipendente di verificatori dei programmi era numeroso circa quanto il gruppo dei programmatori. Per ridurre ulteriormente il rischio degli errori di programma, talvolta si formano due gruppi di programmazione per produrre indipendentemente il software per lo stesso compito. L'ipotesi sottintesa è che almeno una versione del software funzionerà.

La protezione contro gli errori dell'operatore richiede quella che in modo poco gentile viene chiamata «soluzione a prova d'idiota». Una progettazione attenta e accurata del software può ridurre al minimo la probabilità di tali errori. Si possono anche inserire forme di controllo della congruenza di ciò che l'operatore immette: per esempio, il programma può controllare se i valori numerici immessi rientrano nell'intervallo ammesso. Un'altra tecnica è quella di lasciare all'operatore la possibilità di tornare sui propri passi, nel caso di attività cruciali che potrebbero essere avviate per errore. Per esempio, il programma, ricevendo un comando come «Ferma il processo», prima di procedere può chiedere una conferma: «Vuoi veramente che fermi il processo?».

Una proprietà interessante del software è che dopo aver dimostrato che un programma funziona correttamente continuerà a funzionare indefinitamente; il software non si «rompe» con l'uso. Al contrario, un hardware affidabile in un certo momento può non esserlo in un tempo successivo. Gli elaboratori possono fermarsi insieme o calcolare risultati erronei; la memoria e i sensori possono fornire valori scorretti; le linee di comunicazione possono ingarbugliare le informazioni o perderle e gli attuatori possono smettere di funzionare o diventare poco precisi. Potrebbe guastarsi un singolo componente, oppure potrebbe andare fuori uso un intero calcolatore, con la sua memoria e le sue linee di comunicazione, per esempio su un aeroplano dopo un incendio. Anche se il sistema di calcolo stesso è senza pecche, può ancora presentare malfunzionamenti a causa di effetti ambientali come fluttuazioni di tensione o calore eccessivo. Se un sistema deve essere affidabile, perciò, dovrebbe tollerare i malfunzionamenti che ancora sorgono nonostante tutti gli sforzi fatti per eliminarli. La ridondanza sotto molte forme è il mezzo per prevenire che questi malfunzionamenti interferiscano con l'affidabilità pianificata del sistema.

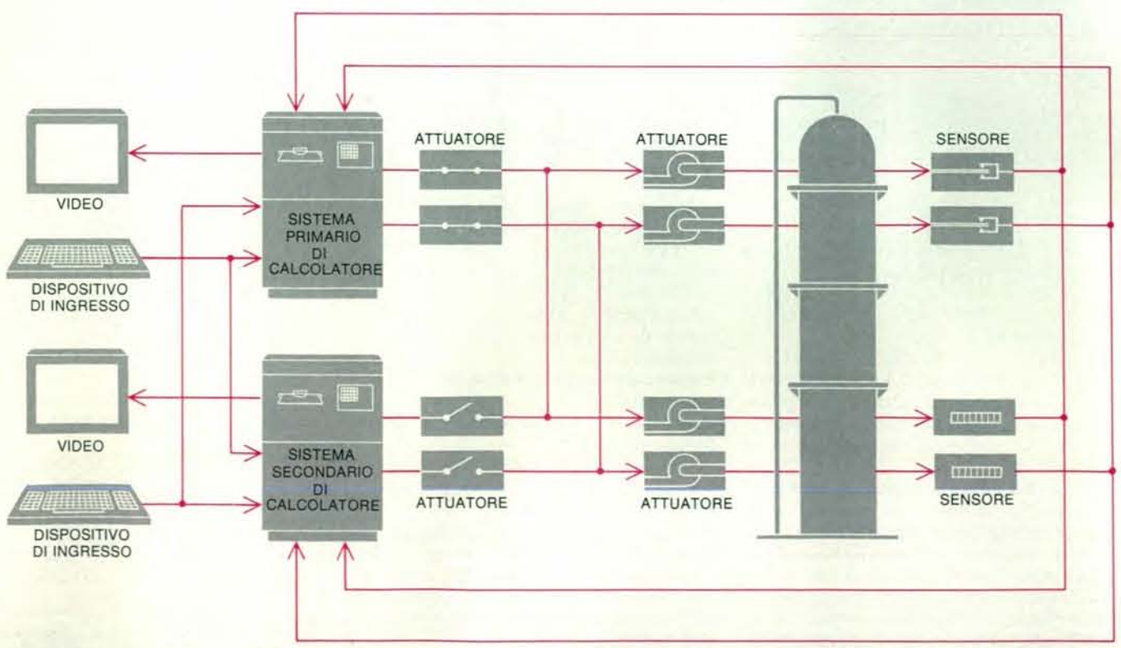
Una certa quantità di ridondanza si può inserire nell'hardware. Per esempio, molti calcolatori immagazzinano informazioni aggiuntive, con ogni elemento in memoria, cosicché alcuni errori di memorizzazione si possono correggere automaticamente. Alcuni elaboratori ritentano automaticamente un'istruzione che ha fallito, accorgimento di progetto fondato sulla ragionevole probabilità che la seconda volta il tentativo avrà successo. Si possono replicare i moduli logici più importanti, persino interi elaboratori. I risultati ottenuti con le unità che operano in parallelo vengono confrontati e il risultato accettato viene determinato con la regola della maggioranza. La tecnica viene chiamata ridondanza modulare n-replicata. Se n è maggiore di 3, la tecnica fornisce un valore corretto se non ci sono più di $(n-1)/2$ malfunzionamenti. Fintanto che i moduli hanno guasti tra loro indipendenti l'affidabilità del sistema cresce al crescere di n.

Per guadagnare in affidabilità si impiegano anche tecniche di software. Come l'hardware, un software può riprovare una procedura dopo un malfunzionamento. Nella maggior parte dei sistemi di comunicazione il software ritrasmette dati finché non ha ricevuto un segnale di ricezione accettata e riconosciuta. Il software, dopo il malfunzionamento di una sorgente, può anche rivolgersi a una sorgente ridondante. In una condizione tipica il software scoprirebbe il malfunzionamento di un attuatore notando incoerenze nei dati provenienti da un sensore; i comandi verrebbero allora indirizzati a un altro attuatore.

In un sistema multicalcolatore si possono eseguire procedure di software analoghe su più elaboratori per attenuare le conseguenze del guasto di un elaboratore. Il software per ciascun elaboratore può essere programmato in modo indipendente per accrescere la possibilità che almeno una versione del software sia corretta. Se il sistema ha elaboratori autonomi alimentati separatamente, distribuiti su più punti distanti e collegati con canali di comunicazione ridondanti, vi sono poche probabilità che l'intero sistema possa guastarsi. Con l'aggiunta di sensori e attuatori ridondanti, un sistema di questo tipo può fornire un'affidabilità estremamente elevata.

Un modo di organizzare un sistema a ridondanza estesa è quello di individuare un calcolatore come primario, considerando gli altri calcolatori come secondari. Il calcolatore primario riceve dati dai sensori e comanda gli attuatori. Anche i calcolatori secondari possono ricevere dati dei sensori per poter disporre del corretto stato del modello, nell'eventualità che uno di essi debba sostituirsi al primario, ma non comandano gli attuatori. I secondari controllano il calcolatore primario periodicamente sia implicitamente, facendo attenzione alla coerenza dei dati provenienti dai sensori, sia esplicitamente, inviando messaggi al primario per chiedergli di eseguire alcune funzioni di verifica. Sia il primario sia i secondari devono filtrare l'informazione dai sensori ridondanti e possono dover «votare» sulle letture fornite dai sensori per assicurarsi che i calcoli si basino su dati validi.

Forse la parte più difficile di questa tecnica è determinare in modo affidabile quando un calcolatore primario è guasto. È facile immaginare una situazione in cui il malfunzionamento di un calcolatore secondario potrebbe usurpare il controllo a un primario correttamente funzionante. Chiaramente è necessario un accordo tra più calcolatori secondari perché almeno uno di essi possa assumere le funzioni del primario. Tutte le volte che c'è tempo a sufficienza è più semplice che sia un operatore a prendere la decisione.



La protezione contro i guasti di sistema è più importante nei sistemi di controllo di processo che in molte altre applicazioni informatiche. Un modo per migliorare l'affidabilità è quello di duplicare i componenti chiave del sistema. Qui un calcolatore è designato come primario e l'altro come secondario. Ciascun calcolatore riceve tutti gli ingressi, ma solo il primario è collegato agli attivatori. Il secondario esegue semplicemente i calcoli come se stesse controllando il processo. Se il primario si guasta, il controllo degli attuatori è passato al secondario. Il sistema è ridondante anche nei sensori, negli attuatori, nei visualizzatori e nei dispositivi di ingresso, perché siano affidabili come i calcolatori.

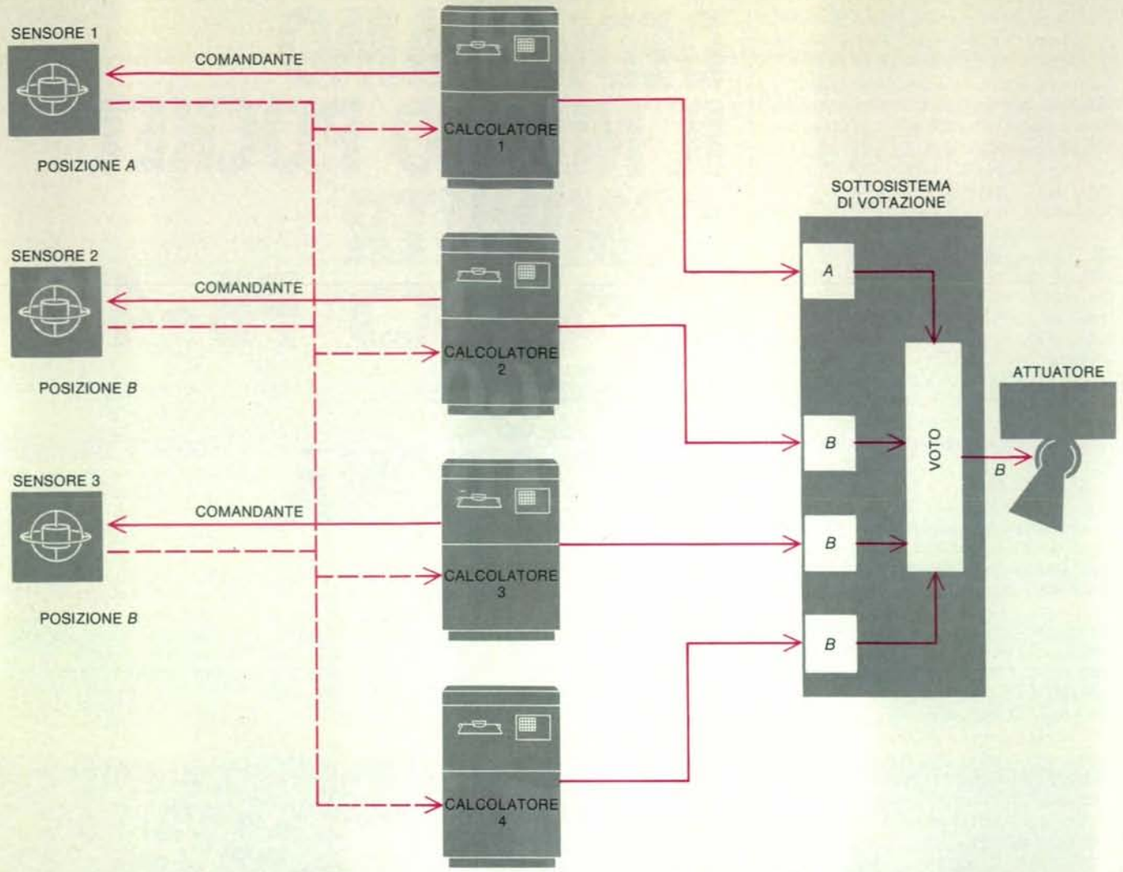
DEDICHIAMO IL NUOVO POCKET COMPUTER CASIO® PB-410 A DIEGO MARADONA.

Ci sarebbe proprio piaciuto ingaggiarlo per la nostra campagna, ma il Napoli ci ha battuti sul filo di lana dei 12 miliardi. Peccato. Il nuovo pocket personal CASIO BP-410, con memoria RAM a schede da 2 a 4 KB intercambiabili e utilizzabili anche come sistema di registrazione dati, è il personal ideale per tutti coloro che non vogliono rinunciare a portare costantemente con sè uno strumento per la gestione d'affari personale. Versatile e capace, con linguaggio di programmazione BASIC, e una Banca Dati per appunti e agenda personale predisposta per l'uso senza necessità di programmazione, lo consigliamo caldamente a Diego Maradona che a soli 23 anni riesce a

muovere attorno a sè tanti miliardi e a risvegliare l'interesse di una città dinamica come Napoli.
Siamo anzi ben disposti a regalargliene uno e a illustrargliene i pregi di persona quando e come vorrà: lo aspettiamo a Milano nostro ospite.
Da parte nostra speriamo che l'effetto Maradona continui e ci auguriamo infatti di vendere tanti PB-410 proprio a Napoli.
In tutti i casi il PB-410 è ovviamente dotato di una facile guida alla programmazione, di una interfaccia per cassette e, per gli affari che possono lasciare traccia di sè, di una utilissima stampante.

CASIO®
Gioielli della microinformatica.

DITRON
DISTRIBUTORE NAZIONALE
SpA
Viale Certosa, 138 Milano - Tel. 02/3085645 (5 linee ric. aut.)



Spesso i sistemi di controllo di processo vengono dotati di una organizzazione «a votazione» per proteggerli contro i guasti. Lo schema di votazione idraulica mostrato è impiegato sulla navetta spaziale per posizionare la sospensione cardanica del razzo. Il sistema ha tre sensori, quattro calcolatori e quattro attuatori idraulici. Solo un calcolatore emette comandi per una lettura da ciascun sensore, mentre gli altri ricevono dati da tutti i sensori. I calcolatori si scambiano informazioni a sufficienza per assicurarsi di essere in accordo sui dati; se non lo sono, scartano i dati. Altrimenti calcolano indipendentemente l'uscita appropriata per mezzo di algoritmi identici. Ciascun calcolatore comanda un attuatore separato; se i comandi sono in conflitto, tre attuatori possono soverchiare il quarto, secondo la regola della maggioranza. Se un componente si guasta, un membro del gruppo può affrontare il problema, come indicano le linee etichettate «comandante».

Un modo alternativo per organizzare un sistema multicalcolatore si basa sulla «votazione». La procedura è analoga alla ridondanza *n*-modulare salvo che la votazione avviene per mezzo di moduli di software e non mediante l'hardware. I moduli eseguono calcoli, scambiano risultati l'uno con l'altro e votano sui risultati. Poiché elaboratori indipendenti elaborano a velocità leggermente differenti, un gruppo dovrà attendere finché l'ultimo non abbia completato i calcoli; poi tutti insieme possono votare.

Un sistema in qualche modo simile a quello descritto controlla i segmenti critici di volo sulla navetta spaziale. Il sistema ha quattro elaboratori autonomi. Ciascun elaboratore comanda un attuatore distinto per ciascuna funzione, come per esempio il movimento delle superfici alari durante la discesa planata nell'atmosfera. La votazione avviene idraulicamente: tre attuatori possono soverchiare un attuatore. Una descrizione degli attuatori idraulici in competizione l'uno contro l'altro può sembrare fuori posto nell'ambito di una discussione sul software, invece è del tutto pertinente. Lo schema di votazione idraulica può esistere grazie al software di controllo di processo il quale, per sua propria natura, è preposto a influenzare direttamente processi di eventi reali. È questo controllo diretto che distingue la maggior parte dei sistemi di controllo di processo da altri sistemi di calcolatori, i quali riportano semplicemente i risultati di calcoli.

Alcuni sistemi di controllo di processo sono inerentemente molto semplici. Il sistema di controllo del riscaldamento di cui ho parlato è un esempio. Altri sistemi, come quelli che controllano la navetta, un aereo a reazione, le centrali telefoniche di commutazione, sono tra i più complessi sistemi automatici mai costruiti. Essi esigono una pianificazione strategica avanzata, alte prestazioni, alta affidabilità e una temporizzazione precisa. Raggiungere questi obiettivi significa usare tecniche mutuate da campi quali l'ingegneria del software, l'architettura dell'hardware, la progettazione di sistemi, i sistemi operativi, l'intelligenza artificiale.

Un sistema semplice si può progettare in tempi dell'ordine di giorni. Un sistema complesso, come il sistema di bordo della navetta spaziale, richiede anni di lavoro per migliaia di addetti. Forse il problema più urgente da risolvere è come ridurre il tempo necessario a costruire un sistema complesso di controllo. È quasi sempre possibile ideare hardware e software adatti al controllo di un processo, ma la realizzazione può essere enormemente costosa. Sebbene certe nuove tecniche di ingegneria del software e certi linguaggi di programmazione (l'Ada è uno di questi) potranno essere in qualche modo d'aiuto, le necessità reali sono quelle di metodi migliori per la stesura di specifiche, di mezzi di analisi e di verifica più facili e di una migliore organizzazione dei sistemi.

Software nella scienza e nella matematica

La computazione è un nuovo strumento per descrivere e studiare sistemi scientifici e matematici: la simulazione al calcolatore può essere l'unico modo per prevedere come certi sistemi complicati si evolvono

di Stephen Wolfram

Le leggi della scienza forniscono algoritmi, o procedure, per determinare il comportamento dei sistemi. Il programma di un calcolatore è un veicolo tramite il quale gli algoritmi possono essere espressi e applicati. In un calcolatore gli oggetti fisici e le strutture matematiche possono essere rappresentati sotto forma di numeri e di simboli e si può scrivere un programma per elaborare questi enti in conformità con gli algoritmi. Quando viene eseguito, il programma provoca nei numeri e nei simboli certe modifiche, dettate dalle leggi scientifiche: è così che esso permette di ricavare le conseguenze di quelle leggi.

Eseguire un programma di calcolatore somiglia molto a effettuare un esperimento. A differenza degli oggetti fisici di un esperimento tradizionale, tuttavia, gli oggetti di un esperimento al calcolatore non sono soggetti alle leggi di natura: essi obbediscono invece alle leggi espresse nel programma, la cui forma, purché coerente, può essere arbitraria. L'impiego del calcolatore estende così i confini della scienza sperimentale, consentendoci di effettuare esperimenti in un universo ipotetico. Il calcolatore amplia anche la scienza teorica: le leggi scientifiche sono state formulate tradizionalmente in termini di una classe particolare di funzioni ed enti matematici e sovente il loro rigoglio è stato favorito non meno dalla loro semplicità matematica che dalla loro attitudine a fornire un modello degli aspetti caratteristici di un fenomeno. Se viceversa una legge scientifica è specificata da un algoritmo, essa può avere una forma arbitraria, purché coerente. Lo studio di molti sistemi complessi, che hanno resistito all'analisi con i metodi tradizionali della matematica, è reso quindi possibile dagli esperimenti e dai modelli al calcolatore. L'impiego di questa macchina si sta rivelando uno strumento nuovo e importantissimo della scienza, che si affianca alle metodologie teoriche e sperimentali di più antica tradizione.

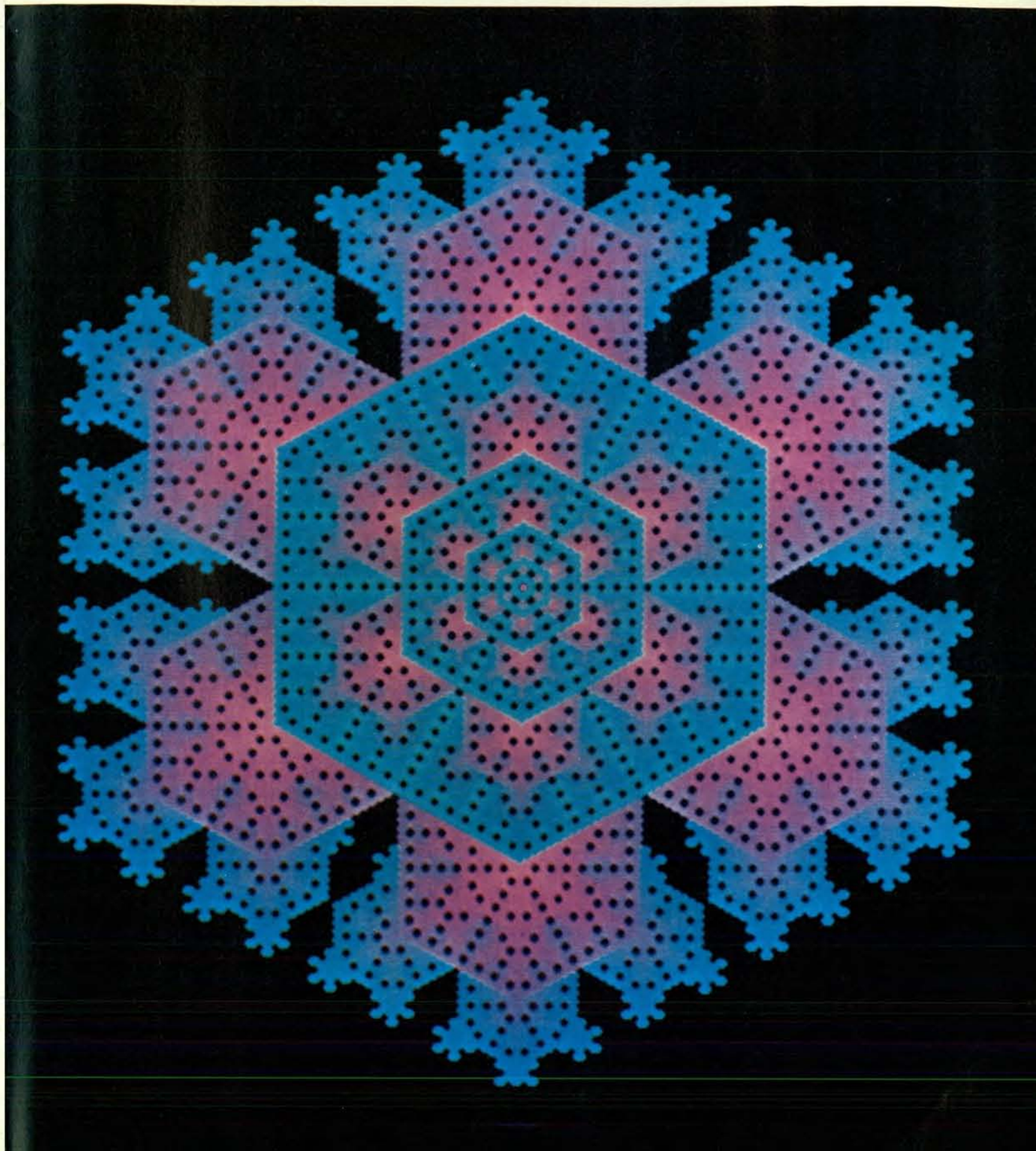
Naturalmente molti calcoli scientifici possono essere eseguiti anche con gli strumenti matematici tradizionali, senza l'ausilio del calcolatore. Ad esempio, assegnate le equazioni che descrivono il moto degli elettroni in un campo magnetico arbitrario, è possibile ricavarne una semplice formula matematica che fornisce la traiettoria di un elettrone in un campo magnetico uniforme (un campo cioè in cui l'intensità è la stessa in tutti i punti). Per campi magnetici più complicati, tuttavia, non esiste una formula matematica altrettanto semplice. Le equazioni del moto forniscono ancora un algoritmo con il quale la traiettoria di un elettrone può essere determinata. In linea di principio questa traiettoria potrebbe essere calcolata a mano, ma in pratica solo un calcolatore è in grado di compiere i numerosissimi passi che occorrono per ottenere risultati precisi.

Un programma di calcolatore che esprima le leggi del moto di un elettrone in un campo magnetico può essere impiegato per eseguire esperimenti al calcolatore. Questi esperimenti sono più flessibili di quelli tradizionali compiuti in laboratorio. Per esempio si può allestire facilmente un esperimento di laboratorio per studiare la traiettoria di un elettrone che si muova soggetto al campo magnetico presente in un cinescopio, ma nessun esperimento di laboratorio potrebbe riprodurre le condizioni in cui si troverebbe un elettrone che si movesse nel campo magnetico che circonda una stella di neutroni. Il programma di calcolatore invece può essere usato sia nel primo sia nel secondo caso.

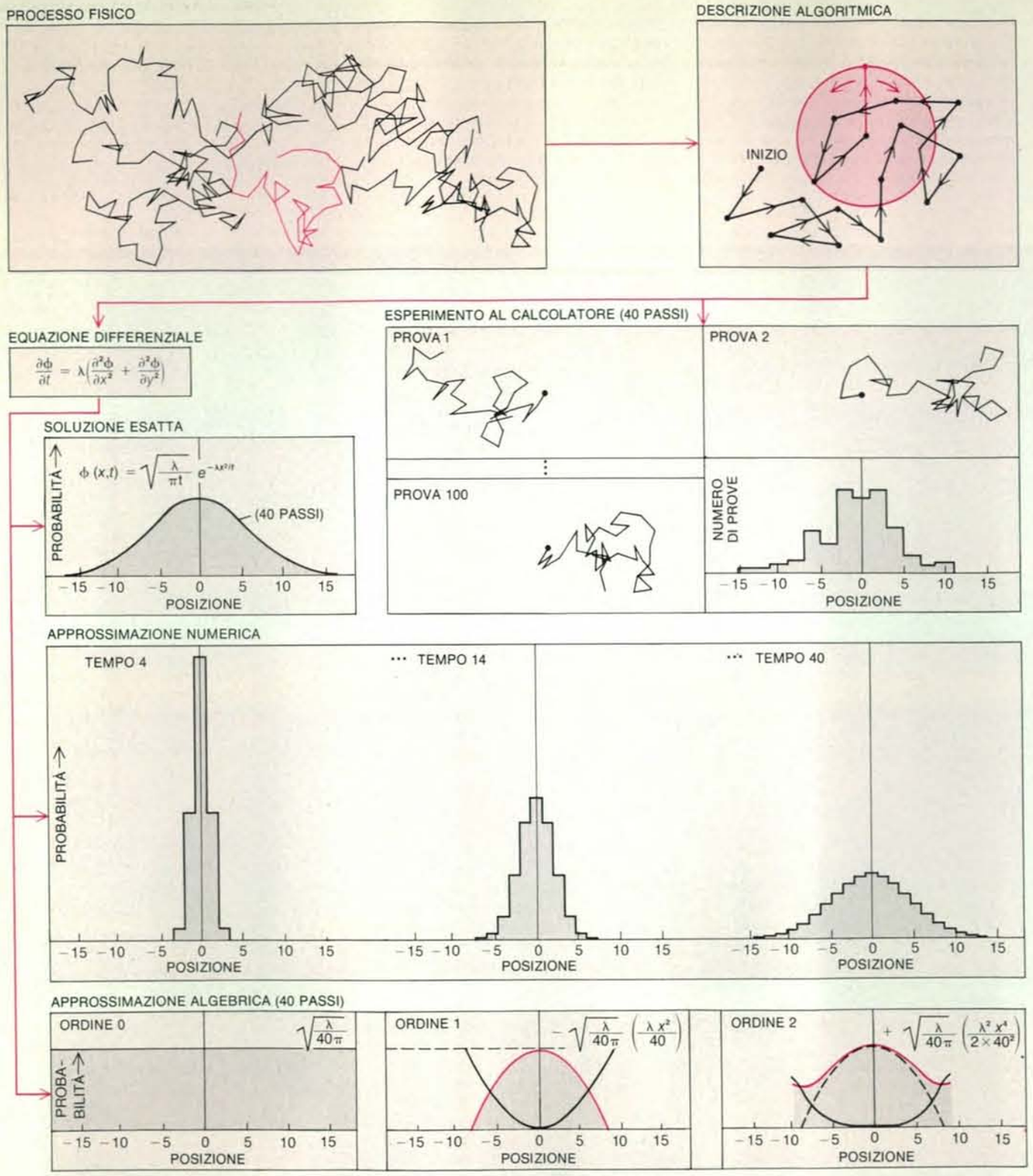
Il campo magnetico studiato è specificato da un insieme di numeri contenuti nella memoria del calcolatore. Il programma applica un algoritmo che simula il moto dell'elettrone modificando via via i numeri che rappresentano la sua posizione in istanti successivi. I calcolatori odierni sono abbastanza veloci da eseguire le simulazioni rapidamente e quindi è possibile studiare un gran numero di casi. Il ricercatore può interagire direttamente con il calcolatore, modificando questo o quell'aspetto del fenomeno a mano a mano che si ottengono nuovi risultati. Con l'ausilio del calcolatore è possibile percorrere molto più rapidamente il ciclo tradizionale del metodo scientifico, in cui le ipotesi vengono prima formulate e poi verificate.

Gli esperimenti al calcolatore non sono limitati ai processi che si presentano in natura. Un programma può, ad esempio, descrivere il moto di monopoli magnetici in un campo magnetico, anche se i monopoli magnetici non sono mai stati osservati in esperimenti fisici. Inoltre, il programma può essere modificato in modo da esprimere svariate leggi di moto per i monopoli magnetici. Anche in questo caso quando il programma viene eseguito si possono determinare le conseguenze di queste leggi ipotetiche. Il calcolatore mette dunque lo studioso in condizione di sperimentare con tutta una gamma di leggi di natura ipotetiche.

Il calcolatore può essere impiegato anche per studiare le proprietà dei sistemi matematici astratti. Gli esperimenti matematici eseguiti al calcolatore sono spesso tali da suggerire delle congetture che in seguito vengono confermate da dimostrazioni matematiche tradizionali. S'immagini di usare un certo modello matematico per studiare la traiettoria di un fascio di elettroni che si muovano immersi nei campi magnetici di un acceleratore di par-



La simulazione al calcolatore ha reso possibile impiegare molti generi nuovi di modelli per i fenomeni naturali. Qui le tappe della formazione di un fiocco di neve sono generate da un programma di calcolo che dà corpo a un modello chiamato automa cellulare. Nel modello il piano è diviso in un reticolo di piccole cellule a forma di esagono regolare; a ciascuna cellula viene assegnato il valore 0, che corrisponde al vapore acqueo *(in nero)*, oppure il valore 1, che corrisponde al ghiaccio *(in colore)*. A cominciare da un'unica cellula rossa al centro dell'illustrazione, il fiocco di neve simulato cresce attraverso una successione di passi. Il valore che assume a ogni passo una cellula situata al bordo del fiocco di neve dipende dal valore totale delle sei cellule che la circondano: se questo valore totale è dispari, la cellula diventa di ghiaccio e assume il valore 1; altrimenti resta di vapore e mantiene il valore 0. Gli strati di ghiaccio successivi che si formano in questo modo sono illustrati con una successione di colori che va dal rosso al blu ogni volta che il numero degli strati raddoppia. Il calcolo necessario relativamente a ciascuna cellula è semplice, ma per la configurazione qui riportata sono occorsi più di 10 000 calcoli. L'unico modo pratico per generare la configurazione è la simulazione al calcolatore. L'illustrazione è stata realizzata con l'ausilio di un programma scritto da Norman H. Packard dell'Institute for Advanced Study.



I metodi matematici e computazionali vengono applicati in vari modi allo studio delle passeggiate aleatorie. La passeggiata aleatoria è un modello per processi fisici come il moto browniano di una minuscola particella sospesa in un liquido. La particella subisce deviazioni casuali dovute al bombardamento delle molecole del liquido; la sua traiettoria può essere quindi descritta come una successione di tratti aventi ciascuno una direzione casuale. Il modo più diretto per ricavare le conseguenze del modello è di ricorrere a un esperimento al calcolatore. Con il calcolatore vengono simulate molte passeggiate aleatorie, poi vengono misurate le loro proprietà medie. Nell'istogramma l'altezza di ogni rettangolo è proporzionale al numero delle passeggiate aleatorie simulate che dopo un certo tempo hanno raggiunto una posizione contenuta in una data area. All'aumentare del numero di prove, la forma dell'istogramma tende a quella della distribuzione esatta delle posizioni. Per una passeggiata aleatoria ordinaria è possibile ricavare la distribuzione esatta direttamente: si può impostare un'equazione differenziale per la distribuzione e l'equazione è abbastanza semplice da consentire una soluzione esatta. In generale tuttavia non è possibile ottenere una soluzione esatta per le equazioni differenziali ed è necessario ricorrere alle approssimazioni. Quando si compie un'approssimazione numerica, la variazione continua delle grandezze che compaiono nell'equazione differenziale viene approssimata da un gran numero di piccoli incrementi. I risultati illustrati sono stati ottenuti mediante un programma in cui gli incrementi spaziali e temporali erano piccole frazioni delle lunghezze e dei tempi corrispondenti ai singoli passi della passeggiata aleatoria. Le approssimazioni algebriche dell'equazione differenziale assumono la forma di una serie di termini algebrici; il contributo di ciascun termine ha l'aspetto di una retta o di una curva continua *(in nero)*. La retta o la curva è sommata per sovrapposizione alla retta o alla curva tratteggiata *(in nero)* che corrisponde all'ordine di approssimazione precedente. Questa sovrapposizione dà come risultato l'ordine di approssimazione corrente *(curve continue in colore)*.

ticelle circolare. Lo spostamento trasversale di un elettrone quando oltrepassa un dato punto in una delle sue rivoluzioni lungo l'anello dell'acceleratore è dato da una certa frazione x compresa tra 0 e 1. Il valore della frazione che corrisponde allo spostamento dell'elettrone nella rivoluzione successiva è allora $ax(1-x)$, dove a è un numero che può variare fra 0 e 4. Questa formula fornisce un algoritmo con il quale si possono ricavare tutti i valori relativi agli spostamenti successivi dell'elettrone.

Bastano poche prove per vedere come le proprietà di questa successione dipendano dal valore di a. Se a è uguale a 2 e il valore iniziale di x è uguale a 0,8, il valore successivo di x, dato da $ax(1-x)$, è uguale a 0,32; applicando un'altra volta la formula si ottiene per x il valore 0,4352; dopo parecchie iterazioni la successione dei valori di x converge a 0,5. In effetti, se a è piccolo e x è un numero arbitrario compreso fra 0 e 1, la successione si stabilizza rapidamente e fornisce lo stesso valore di x a ogni rivoluzione dell'elettrone.

Quando a aumenta, tuttavia, si può osservare un fenomeno detto raddoppio del periodo. Quando a raggiunge il valore 3, la successione comincia a oscillare fra due valori di x. A mano a mano a continua ad aumentare, prima compaiono quattro valori di x, poi otto e infine, quando a raggiunge circa il valore 3,57, ne compare tutta una gamma. Non sarebbe facile prevedere questo comportamento dalla costruzione del modello matematico, ma l'esperimento al calcolatore lo suggerisce immediatamente. Le proprietà del modello possono essere poi determinate in ogni particolare mediante una dimostrazione tradizionale.

I processi matematici che possono esser descritti da un programma di calcolatore non sono solo quelli corrispondenti alle operazioni e alle funzioni della matematica tradizionale. Per esempio non esiste nella matematica tradizionale un simbolo indicante la funzione che inverte l'ordine delle cifre di un numero; nondimeno è possibile definire e applicare questa funzione in un programma. Con il calcolatore è diventato agevole introdurre leggi scientifiche e matematiche la cui natura intrinseca sia algoritmica. Si consideri la catena di eventi che si mette in moto quando un elettrone, accelerato fino ad assumere un'energia molto elevata, viene lanciato contro un blocco di piombo. Vi è una certa probabilità che l'elettrone emetta un fotone avente una data energia. Se il fotone viene emesso, vi è una certa probabilità che esso generi un secondo elettrone e un positone (l'antiparticella dell'elettrone). Ciascun elemento della coppia può a sua volta emettere altri fotoni, sicché alla fine si produce una cascata di particelle. Non esiste alcuna formula matematica semplice che possa descrivere questo processo, anzi neppure i suoi elementi. Nondimeno è possibile inserire in un programma un algoritmo che rappresenti questo processo e l'esito del processo può essere dedotto eseguendo il programma. L'algoritmo funge da legge fondamentale per la descrizione del processo.

Lo strumento matematico fondamentale per costruire la maggior parte dei tradizionali modelli di fenomeni naturali è costituito dalle equazioni differenziali; queste equazioni esprimono le relazioni esistenti fra certe grandezze e il loro tasso di variazione. Per esempio una reazione chimica procede a una velocità che è proporzionale alla concentrazione delle sostanze reagenti e questa relazione può essere espressa mediante un'equazione differenziale; una soluzione di questa equazione fornirebbe la concentrazione di ogni reagente in funzione del tempo. In alcuni casi semplici è possibile trovare una soluzione completa dell'equazione in termini di funzioni matematiche ordinarie, tuttavia, nella maggior parte dei casi, non è possibile ottenere una soluzione esatta e si deve far ricorso a una soluzione approssimata.

Le approssimazioni più comuni sono quelle numeriche. Supponiamo che uno dei termini dell'equazione differenziale fornisca la velocità di variazione istantanea di una grandezza in funzione del tempo. Questo termine può essere approssimato dalla variazione totale di quella grandezza in un breve intervallo, variazione che va poi inserita nell'equazione differenziale. L'equazione che così si ottiene è in realtà un algoritmo che determina il valore approssimato della grandezza alla fine di un intervallo, noto il suo valore all'inizio di quell'intervallo. Applicando ripetutamente l'algoritmo a intervalli successivi, è possibile trovare la variazione approssimata della grandezza in funzione del tempo e i risultati sono tanto più precisi quanto più brevi sono gli intervalli. I calcoli necessari per ciascun intervallo sono semplicissimi, ma nella maggior parte dei casi devono essere ripetuti molte volte per ottenere un grado di precisione accettabile. Un metodo del genere è dunque praticabile solo al calcolatore.

I metodi numerici tradotti in programmi di calcolatore sono stati impiegati per trovare soluzioni approssimate delle equazioni differenziali che si incontrano in un'ampia gamma di discipline. In certi casi le soluzioni hanno una forma semplice; in molti altri, tuttavia, esse manifestano un comportamento complicato, quasi casuale, anche se le equazioni differenziali corrispondenti sono semplicissime. In questi casi è necessario utilizzare la matematica sperimentale.

Nelle applicazioni pratiche non solo si trova che spesso le equazioni differenziali sono complicate, ma anche che ne esistono molte. Per esempio i modelli teorici delle esplosioni nucleari impiegati nella progettazione di armi e lo studio delle supernove implicano centinaia di equazioni differenziali che descrivono le interazioni di molti isotopi. In pratica questi modelli sono sempre usati sotto forma di programmi di calcolatore, poiché solo

Collezione Biblioteca

Per chi conserva il gusto della lettura

Massimo Mila
WOLFGANG AMADEUS MOZART
pp. 181
Edizione con astuccio
L. 10.000

Gustav Meyrink
LA NOTTE DI VALPURGA
Prefazione di Marino Freschi
pp. XXVII - 200
Edizione con astuccio
L. 14.000

Andrea Della Corte
ARTURO TOSCANINI
pp. 471
Edizione con astuccio
L. 22.000

Arthur Schnitzler
LA DANZATRICE GRECA
Presentazione di Italo Alighiero Chiusano
pp. XXVI - 116
Edizione con astuccio
L. 10.000

Massimo Mila
I COSTUMI DELLA TRAVIATA
pp. 336
Edizione con astuccio
L. 25.000

Elena Vetsera
MAYERLING
Con un saggio di Italo Zingarelli
pp. XXXVII - 82
Edizione con astuccio
L. 10.000

Vito Levi
RICHARD STRAUSS
pp. XXIV - 148
Edizione con astuccio
L. 16.000

Giorgio Voghera
NOSTRA SIGNORA MORTE
pp. 150
Edizione con astuccio
L. 12.000

Italo Alighiero Chiusano
GOETHIANA
pp. XIX - 143 Ill.
Edizione con astuccio
L. 10.000

Thomas Mann
IL BAMBINO PRODIGIO
e altri racconti
pp. XXX - 134
Edizione con astuccio
L. 14.000

Autori Vari
WILLIAM BLAKE MITO E LINGUAGGIO
a cura di Giovanna Franci
pp. XVII - 204 Ill.
Edizione con astuccio
L. 15.000

Franz Werfel
IL SEGRETO DI UN UOMO
Prefazione di Giorgio Cusatelli
pp. XXXIV - 148
Edizione con astuccio
L. 10.000

Rainer Maria Rilke
DUE RACCONTI PRAGHESI
pp. XXVII - 125
Edizione con astuccio
L. 10.000

Miroslav Krleža
SULL'ORLO DELLA RAGIONE
A cura di Silvio Ferrari
pp. X - 264
Edizione con astuccio
L. 20.000

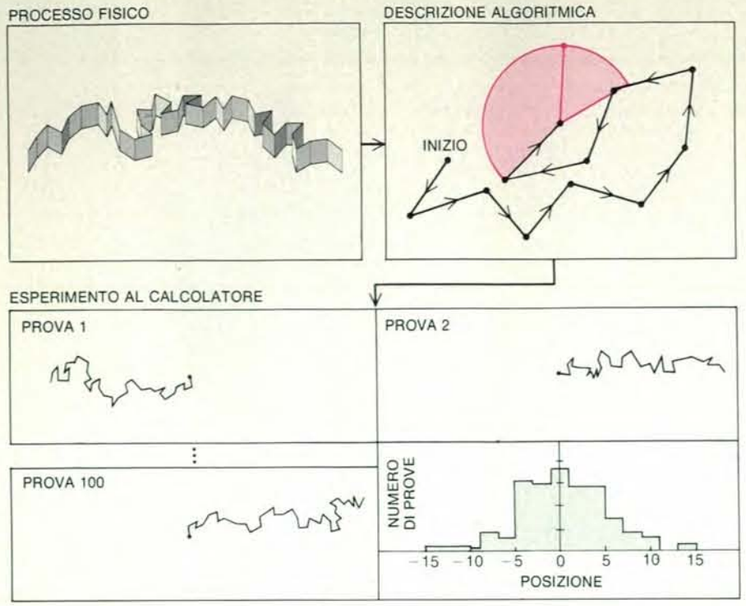
Martin Lutero
LE 95 TESI
Presentazione di Sergio Quinzio
pp. XX - 172
Edizione con astuccio
L. 16.000

John William Polidori
IL VAMPIRO
pp. XXXII - 128
L. 15.000

Gian Francesco Malipiero
IGOR STRAVINSKIJ
Presentazione di Gianandrea Gavazzeni
pp. XXVI - 80
Edizione con astuccio
L. 10.000

William Blake
CANTI DELL'INNOCENZA E DELL'ESPERIENZA
pp. 160
Edizione con astuccio
L. 16.000

Edizioni Studio Tesi



I metodi computazionali sono usati, da soli, per lo studio delle passeggiate aleatorie senza incroci. Queste passeggiate, che possono essere utilizzate come modelli di processi fisici come il ripiegamento di molecole polimeriche, differiscono dalle passeggiate aleatorie ordinarie in quanto ciascun passo deve evitare tutti i passi precedenti. Questa complicazione rende impossibile costruire una semplice equazione differenziale che descriva le proprietà medie della passeggiata. Per questo motivo le impostazioni matematiche classiche non servono. Le proprietà delle passeggiate aleatorie senza incroci vengono scoperte ricorrendo alla simulazione diretta al calcolatore.

una macchina è in grado di seguire le interrelazioni fra un numero così elevato di grandezze.

I risultati di certi calcoli numerici, come per esempio l'abbondanza di elio nell'universo, possono essere espressi con numeri singoli. Per lo più, tuttavia, si è interessati alla variazione di certe grandezze al variare dei parametri del calcolo; quando i parametri sono solo uno o due i risultati possono essere illustrati mediante un grafico, ma quando vi siano più di due parametri, sovente i risultati non possono essere espressi in modo conciso se non mediante una formula matematica. Di solito non si riesce a trovare una formula esatta, ma spesso è possibile ottenere delle formule approssimate. Queste formule sono molto comode perché, a differenza dei grafi e delle tavole numeriche, possono essere introdotte direttamente in altri calcoli.

È frequente che la formula approssimata si presenti come una serie di termini, ciascuno dei quali contiene una variabile elevata a qualche potenza, il cui esponente cresce da un termine al successivo. Quando la variabile ha valori piccoli, i termini della serie diventano via via più piccoli; quindi per valori piccoli di x la somma dei primi termini di una serie tipo $1 - x + x^2 - x^3 + \ldots$ fornisce una buona approssimazione della somma di tutta la serie, che è di $1/(1 + x)$. I primi termini di una serie di solito sono facili da calcolare, ma la loro complessità cresce rapidamente; se si vogliono quindi calcolare termini che contengono potenze di x elevate, l'impiego del calcolatore diventa indispensabile.

In linea di principio i programmi di calcolatore possono essere applicati a qualunque ente matematico ben definito, ma in pratica i tipi di enti che possono essere usati in un programma particolare sono in gran parte determinati dal linguaggio in cui il programma è scritto. I metodi numerici richiedono soltanto un insieme limitato di enti matematici e i programmi che esprimono questi metodi possono essere scritti in linguaggi universali come il C, il FORTRAN e il BASIC. Quando si devono ricavare e trasformare delle formule, si devono compiere delle operazioni su enti matematici di livello superiore, per esempio espressioni algebriche, e quindi occorrono linguaggi di programmazione nuovi. Fra i linguaggi di questo genere usati attualmente vi è l'SMP, che è stato ideato dall'autore.

L'SMP è un linguaggio per operare sui simboli; esso non opera solo sui numeri, ma anche su espressioni simboliche che possono rappresentare formule matematiche. Per esempio nell'SMP l'espressione algebrica $2x - 3y + 5x - y$ verrebbe scritta più semplicemente $7x - 4y$. Questa trasformazione è generale e vale per tutti i possibili valori numerici di x e y. Fra le istruzioni fondamentali dell'SMP si trovano le operazioni consuete dell'algebra e dell'analisi matematica *(si veda l'illustrazione a pagina 148)*.

Il linguaggio SMP comprende anche operazioni che permettono di definire e trattare enti matematici di livello superiore in modo assai simile a quello tipico dell'attività matematica ordinaria. Nell'SMP hanno importanza fondamentale sia i numeri reali (che comprendono i razionali e gli irrazionali) sia i numeri complessi (che possiedono una parte reale e una parte immaginaria). Gli enti matematici chiamati quaternioni, che sono un'estensione dei numeri complessi, non sono invece fondamentali; tuttavia anch'essi possono venir definiti all'interno del linguaggio SMP e si possono assegnare regole che definiscono le operazioni di somma e prodotto su di essi. In tal modo le nozioni matematiche dell'SMP possono essere estese.

Alcuni dei vantaggi offerti da un linguaggio come l'SMP si possono paragonare a quelli che si hanno se si usa un calcolatore invece delle tavole logaritmiche. Ormai l'ampia diffusione delle calcolatrici e dei calcolatori ha reso antiquate queste tavole: per ricavare un logaritmo è molto più conveniente ricorrere a un algoritmo per calcolatore che non cercare il risultato sulle tavole. Allo stesso modo, sfruttando un linguaggio come l'SMP è ormai possibile tradurre in forma algoritmica tutto il dominio delle conoscenze matematiche. Per esempio il calcolo degli integrali, che tradizionalmente si esegue con l'ausilio di tabelle, può essere affidato in misura crescente al calcolatore. Quest'ultimo non solo porta a compimento i calcoli con grande velocità e senza errori, ma rende anche automatico il compito di identificare le formule e i metodi appropriati.

Nell'SMP viene riunita una famiglia sempre più vasta di definizioni, allo scopo di far fronte a un'ampia varietà di calcoli matematici. Ormai si può trovare nell'SMP la definizione di varianza statistica, definizione che si può subito applicare per calcolare la varianza nei vari casi particolari. Definizioni siffatte permettono ai programmi scritti nel linguaggio SMP di sfruttare nozioni matematiche sempre più raffinate.

Le equazioni differenziali forniscono modelli adeguati per le proprietà globali di processi fisici come le reazioni chimiche. Esse descrivono per esempio le variazioni della concentrazione totale di molecole; non descrivono invece il moto delle singole molecole. Questi moti possono essere considerati come passeggiate aleatorie: il percorso di ciascuna molecola è simile al percorso di una persona in mezzo a una folla. Nella versione più semplice del modello, si suppone che la molecola si sposti in linea retta finché entra in collisione con un'altra molecola: a questo punto essa riparte in una direzione aleatoria. Si suppone inoltre che tutti i tratti rettilinei siano della stessa lunghezza. Risulta che, se un gran numero di molecole compie passeggiate aleatorie, la variazione media della concentrazione di molecole in funzione del tempo può in effetti essere descritta da un'equazione differenziale, che si chiama equazione di diffusione.

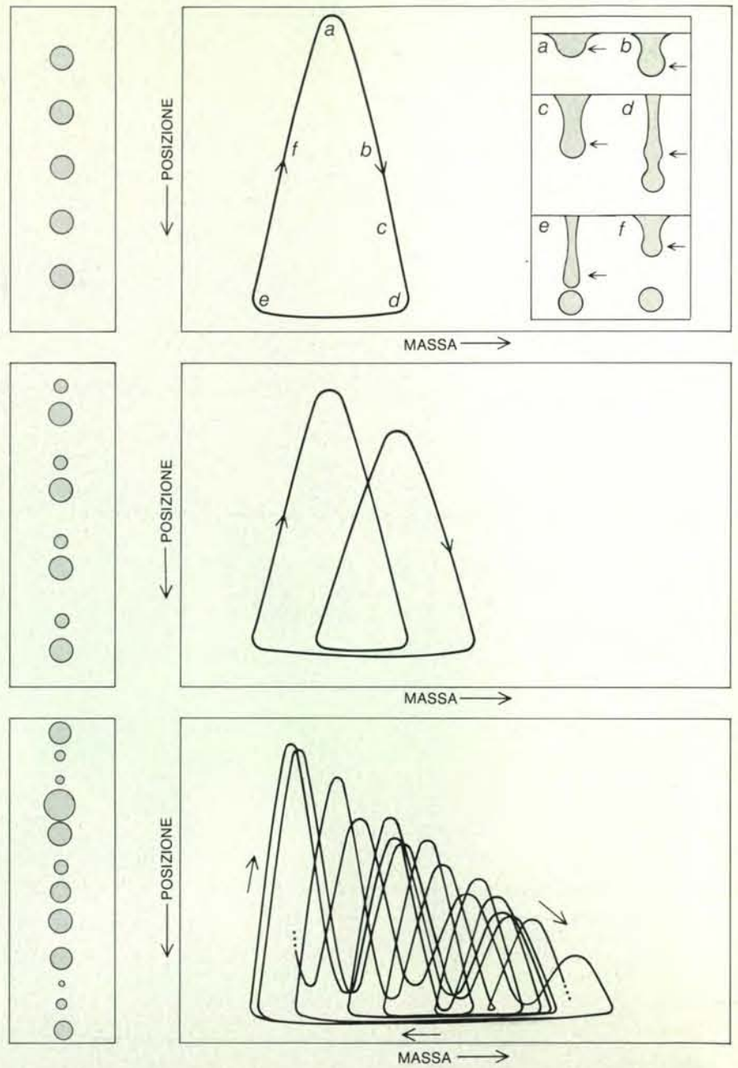
Tuttavia vi sono molti processi fisici per i quali una siffatta descrizione media non appare possibile: in questi casi non sono disponibili equazioni differenziali e si deve ricorrere alla simulazione diretta. È necessario seguire esplicitamente i moti di moltissime molecole o componenti individuali; il comportamento globale del sistema viene stimato trovando le proprietà medie dei risultati. L'unico modo possibile in pratica per compiere queste simulazioni è quello degli esperimenti al calcolatore: senza calcolatore sarebbe sostanzialmente impossibile compiere l'analisi dei sistemi per i quali l'analisi è necessaria.

La passeggiata aleatoria senza incroci è un esempio dei processi che secondo ogni evidenza possono essere studiati soltanto tramite la simulazione diretta. Può essere descritta da un semplice algoritmo che assomiglia alla passeggiata aleatoria ordinaria, ma che ne differisce in quanto i passi successivi della passeggiata aleatoria senza incroci non devono attraversare il percorso compiuto nei passi precedenti. Il modo in cui si ripiegano certe molecole molto lunghe, per esempio il DNA, può essere assimilato a una passeggiata aleatoria senza incroci.

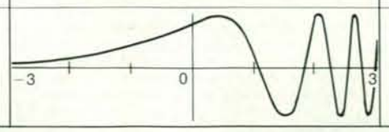
Basta l'introduzione di quest'unico vincolo a rendere la passeggiata aleatoria senza incroci molto più complicata di quella ordinaria. E di fatto per la passeggiata aleatoria senza incroci non si conosce alcuna semplice descrizione media analoga all'equazione di diffusione. Se si vogliono studiare le sue proprietà sembra proprio che non ci sia altro da fare che compiere un esperimento diretto al calcolatore. Il procedimento seguito consiste nel generare un gran numero di campioni di passeggiate aleatorie, scegliendo a ogni passo una direzione casuale; poi delle proprietà di tutte queste passeggiate si fa la media. Questo procedimento è un esempio del metodo di Monte Carlo, così chiamato perché la sua applicazione dipende da fattori aleatori.

Sono stati trovati parecchi esempi di sistemi la cui costruzione è affatto semplice, ma che hanno un comportamento complicatissimo. Lo studio di questi sistemi sta dando origine a una nuova disciplina, la teoria dei sistemi complessi, in cui il metodo computazionale ha una parte fondamentale. L'esempio più classico è la turbolenza dei fluidi, che si osserva per esempio quando l'acqua scorre velocemente intorno a un ostacolo. È facile scrivere il sistema di equazioni differenziali cui obbedisce il fluido e tuttavia ricavare le configurazioni del flusso cui portano le equazioni presenta difficoltà formidabili di descrizione o di analisi matematica. In pratica queste configurazioni vengono trovate o tramite l'osservazione diretta del sistema fisico o, per quanto possibile, mediante esperimenti al calcolatore.

Si pensa che esista un insieme di meccanismi matematici comuni a molti sistemi che danno origine a un comportamento complicato. Questi meccanismi possono essere studiati tanto più agevolmente quanto più semplice è la costituzione del sistema. Uno studio di questo genere è stato condotto di recente su una classe di sistemi matematici noti con il nome di automi cellulari. Un automa cellulare è costituito da molte componenti identiche, ciascuna delle quali si evolve secondo un insieme piuttosto semplice di regole. Pre-



Un comportamento caotico si osserva in molti sistemi naturali. Un esempio familiare è il rubinetto che gocciola, descritto da un modello matematico che è stato formulato in termini di un'equazione differenziale da Robert Shaw, dell'Institute for Advanced Study. Quando l'acqua fluisce attraverso il rubinetto con una portata molto bassa, a intervalli regolari si formano gocce tutte della stessa grandezza *(a sinistra)*. Il modello implica che se si riporta la posizione del culmine di ciascuna goccia che si forma *(frecce)* in funzione della massa della goccia, si ottiene una curva chiusa semplice, chiamata ciclo limite *(a destra)*. L'evoluzione del sistema è rappresentata da un punto che percorre questa curva nel tempo. Se la portata aumenta, all'improvviso il comportamento del sistema diviene più complicato: si presenta un fenomeno chiamato raddoppio del periodo e in ogni ciclo si formano coppie di gocce, spesso di dimensioni diverse. Se la portata aumenta ancora, vi è una successione di ulteriori raddoppi del periodo. Infine, proprio prima che il flusso d'acqua dal rubinetto divenga continuo, si osserva un flusso irregolare di gocce; le gocce hanno tutta una gamma di dimensioni e gli intervalli tra la formazione di una goccia e di quella successiva appaiono casuali. Il comportamento del sistema in questa fase è descritto da una curva irregolare chiamata attrattore strano o caotico. La forma della curva è implicita nell'equazione differenziale, ma in pratica la si può trovare solo con tecniche di approssimazione numerica.

INGRESSO	USCITA	COMMENTO
6+17	23	Calcolare un'espressione numerica.
6/7+8/9	110/63	Calcolare un'espressione numerica con frazioni.
2x−3x+1	1−x	Semplificare un'espressione algebrica.
Ex[(x−1) (x+1)]	$-1+x^2$	Sviluppare espressioni algebriche di prodotti di termini. La notazione x^y significa x elevato a y. Uno spazio fra due espressioni non numeriche sta per moltiplicazione.
Ex[(x−a)^2 (x+2a)^5]	$8a\ x^6+21a^2\ x^5+10a^3\ x^4-40a^4\ x^3-48a^5\ x^2+16a^6\ x+32a^7+x^7$	
Fac[x^2−1]	(−1+x) (1+x)	Scomporre in fattori espressioni algebriche
Fac[x^6−6x^4+4x^3+9x^2−12x+4]	$(-1+x)^4\ (2+x)^2$	
Sol[x^2−3x+1=0,x]	$\{x\to\frac{3-5^{1/2}}{2},\ x\to\frac{3+5^{1/2}}{2}\}$	Risolvere un'equazione nella variabile x.
Sol[{x+3a y=4,y−15x=6b},{x,y}]	$\{x\to\frac{4}{1+45a}-\frac{18a\ b}{1+45a},\ y\to\frac{60}{1+45a}+\frac{6b}{1+45a}\}$	Risolvere un sistema di due equazioni nelle variabili x e y.
Ps[(1+x^3) E^x,x,0,6]	$1+x+\frac{x^2}{2}+\frac{7x^3}{6}+\frac{25x^4}{24}+\frac{61x^5}{120}+\frac{121x^6}{720}$	Trovare una serie di potenze che approssimi $e^x(1+x)^3$ per x prossimo a 0, conservando i termini fino alla sesta potenza di x.
t:x−2a t^2−2t+1	$1+4a-2x+(-2a+x)^2$	Assegnare il valore $x-2a$ al simbolo t; semplificare l'espressione t^2-2t+1 per questo valore di t.
f[2]:6x+1 f[3]:4−x a f[2]+b f[3]+c f[1]	a (1+6x)+b (4−x)+c f[1]	Assegnare il valore $6x+1$ a $f[2]$ e il valore $4-x$ a $f[3]$; calcolare un'espressione in $f[1]$, $f[2]$ e $F[3]$, dove $f[1]$ non è ancora specificata.
f	{[2]:1+6x,[3]:4−x}	Scrivere l'oggetto f, che è un elenco i cui elementi hanno per indici i numeri che compaiono nelle parentesi.
f[1]:7 f	{7,1+6x,4−x}	Assegnare il valore 7 a $f[1]$; scrivere l'oggetto f, che ora è assegnato in forma di vettore, cioè di lista ordinata di elementi.
f^2−8	$\{41,-8+(1+6x)^2,-8+(4-x)^2\}$	Calcolare il quadrato e poi sottrarre 8 da ciascun elemento del vettore f; il risultato è ancora un vettore.
f[p]:5x f[p^2]:6x f	$\{[p^2]:6x,[p]:5x,[1]:7,[2]:1+6x,[3]:4-x\}$	Assegnare dei valori agli elementi di f che hanno indici non numerici; scrivere l'oggetto f.
f[$x]:$x^2 f	$\{[p^2]:6x,[p]:5x,[1]:7,[2]:1+6x,[3]:4-x,[\$x]:\$x^2\}$	Assegnare un valore a $f[\$x]$, dove $\$x$ è un'espressione qualunque; la definizione generale viene collocata alla fine della lista f ed è usata solo quando non si applica nessuno dei casi precedenti. Scrivere l'oggetto f.
f[p]+f[2]+f[a]	$1+11x+a^2$	Calcolare l'espressione $f[p]+f[2]+f[a]$; per calcolare $f[a]$ si applica la definizione generale di $f[\$x]$.
g[$x_ =Natp[$x]]:$x g[$x−1] g[1]:1 g	{[1]:1,[$x_ =Natp[$x]]:$x g[$x−1]}	Definire la funzione fattoriale $g[x]$ per i numeri naturali x, dove $g[N]$ è data da $1\times 2\times\ \ldots\ \times N$. La definizione è data da una formula ricorsiva in cui $g[x]$ è specificata in termini di $g[\$x-1]$. L'espressione $\$x_ = Natp [$x] indica che $\$x$ dev'essere un numero naturale.
g[5]	120	Calcolare $g[5]$, il fattoriale di 5.
Abs[3] Abs[−3] Abs[−x]	3 3 Abs[x]	Trovare il valore assoluto di −3,3 e $-x$.
Abs[$x $$x]:Abs[$x] Abs[$$x]		Definire il valore assoluto del prodotto di due espressioni arbitrarie $\$x$ e $\$\$x$ come il prodotto dei loro valori assoluti.
Abs[$x^($n_ =Natp[$n])]:Abs[$x]^$n		Definire il valore assoluto dell'espressione arbitraria $\$x$ elevata alla potenza $\$n$ (numero naturale) come il valore assoluto di $\$x$ alla $\$n$.
Abs[a b^2 c]	$\mathrm{Abs}[a]\ \mathrm{Abs}[b]^2\ \mathrm{Abs}[c]$	Trovare il valore assoluto di $a\times b^2\times c$ in base alle regole ordinarie dell'algebra e alle definizioni date per la funzione valore assoluto.
Graph[Sin[E^x],x,−3,3]		Tracciare un grafico della funzione $\sin(e^x)$ per i valori di x compresi fra −3 e 3.

Calcoli matematici vengono eseguiti da un calcolatore in questo esempio di dialogo che si svolge nel linguaggio SMP, ideato dall'autore. Il calcolatore può elaborare sia formule algebriche e altri enti simbolici sia numeri. I comandi di questo linguaggio comprendono tutte le operazioni della matematica classica. Gli ultimi riquadri mostrano come si possano definire operazioni nuove. Le proprietà della funzione valore assoluto vengono prima definite e successivamente applicate dal calcolatore per semplificare le espressioni che contengono quella funzione.

se insieme, peraltro, le componenti danno luogo a comportamenti il cui grado di complessità è sostanzialmente arbitrario.

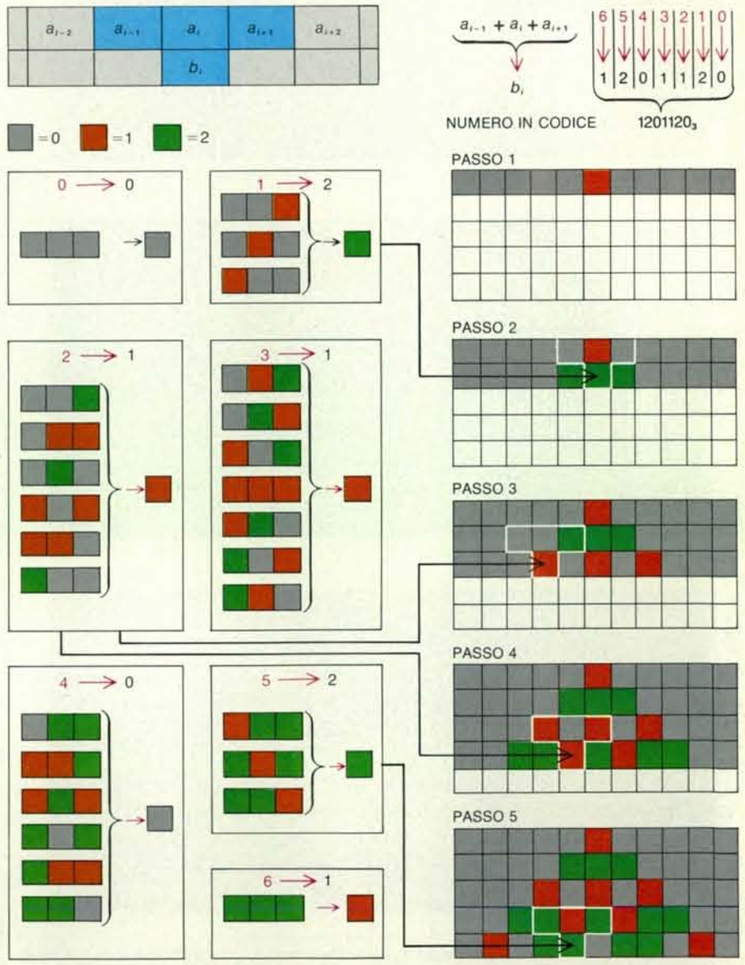
Le componenti di un automa cellulare sono «cellule» matematiche distribuite, in una dimensione, su una successione di punti equidistanti disposti su una linea oppure, in due dimensioni, su un reticolo regolare di quadrati o di esagoni. A ciascuna cellula è associato un valore, scelto tra pochi valori possibili, spesso solo 0 e 1. A ogni scatto di un orologio i valori di tutte le cellule dell'automa vengono aggiornati simultaneamente, secondo una regola ben definita. Questa regola fornisce il nuovo valore di una cellula dato il suo valore precedente e i valori precedenti delle cellule limitrofe o delle cellule di qualche altro insieme vicino.

Si consideri un automa cellulare unidimensionale in cui ogni cellula possa avere i valori 0 o 1. Perfino in questo caso così semplice il comportamento globale dell'automa cellulare può essere molto complesso; il modo più efficace per studiarne il comportamento è quello di ricorrere a un esperimento al calcolatore. E, di fatto, gran parte delle proprietà degli automi cellulari sono state ipotizzate sulla base delle configurazioni osservate negli esperimenti al calcolatore; in alcuni casi queste proprietà sono state confermate successivamente con l'aiuto di dimostrazioni matematiche tradizionali.

Gli automi cellulari possono fungere da modelli espliciti per un'ampia categoria di processi fisici. Supponiamo che su un reticolo esagonale bidimensionale il ghiaccio sia rappresentato da cellule con il valore 1 e il vapore acqueo da cellule con il valore 0. Si può allora impiegare una regola tipo automa cellulare per simulare gli stadi successivi del congelamento di un fiocco di neve: la regola dice che, una volta congelata, una cellula non torna più allo stato liquido. Le cellule raggiunte dal bordo della configurazione di ghiaccio in crescita congelano, a meno che non abbiano tante vicine già congelate da non riuscire a dissipare il calore necessario per congelare. I fiocchi di neve nati in un esperimento al calcolatore da un'unica cellula congelata secondo questa regola posseggono intricate strutture arborescenti che assomigliano parecchio a quelle dei fiocchi di neve reali. Anche un sistema di equazioni differenziali è in grado di descrivere lo sviluppo dei fiocchi di neve, ma il modello molto più semplice fornito dall'automa cellulare pare conservi l'essenza del processo con cui vengono create queste configurazioni complesse. Sembra che modelli analoghi funzionino bene per i sistemi biologici: intricati schemi di crescita e di pigmentazione possono essere interpretati dai semplici algoritmi che generano gli automi cellulari.

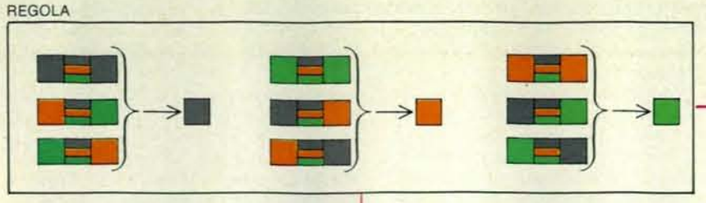
La simulazione al calcolatore è l'unico metodo oggi usato per studiare molti dei sistemi discussi finora. È naturale domandarsi se la simulazione sia, in linea di principio, il procedimento più efficiente possibile o se esista una formula matematica che potrebbe condurre ai risultati in modo più diretto. Per affrontare questo problema è necessario studiare più da vicino la corrispondenza tra i processi fisici e i processi computazionali.

È presumibile che qualsiasi processo fisico possa essere descritto da un algoritmo e che quindi possa essere rappresentato da un processo computazionale. È necessario stabilire quanto sia complicato quest'ultimo processo. Negli automi cellulari la corrispondenza tra processi fisici e computazionali è particolarmente evidente: un automa cellulare può essere considerato come il modello di un sistema fisico, ma può essere considerato anche come un sistema computazionale strettamente affine a un calcolatore digitale ordinario. La successione dei valori presenti all'inizio nelle cellule di un automa cellulare può essere considerata come un insieme di dati o informazioni astratte, proprio come la successione delle cifre binarie nella memoria di un calcolatore digitale. Nel corso dell'evoluzione di un automa cellulare le informazioni vengono elaborate, cioè i valori delle cellule vengono modificati secondo regole ben definite. Analogamente, le cifre immagazzinate nella memoria del calcolatore digitale vengono modificate da regole che sono contenute nell'unità centrale di elaborazione del calcolatore.

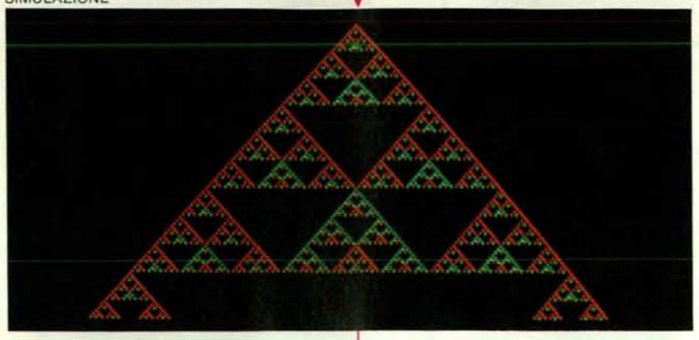


Gli automi cellulari sono modelli semplici che sembrano cogliere le caratteristiche essenziali di un'ampia varietà di sistemi naturali. Un automa cellulare unidimensionale è costituito da una fila di cellule, che nello schema sono rappresentate da quadrati in colore; ciascuna cellula può assumere un certo numero di valori, rappresentati da colori diversi. L'automa cellulare si evolve in una serie di passi, illustrati da una successione di righe di quadrati che va dall'alto verso il basso della pagina. A ogni passo i valori di tutte le cellule vengono aggiornati secondo una regola fissa. Nel caso qui illustrato la regola definisce il nuovo valore di una cellula in termini della somma del suo valore precedente e dei valori precedenti delle cellule a essa immediatamente vicine. È conveniente specificare regole di tal genere mediante numeri in codice definiti come appare nell'illustrazione; il 3 posto a indice è necessario dato che ciascuna cellula può assumere uno di tre valori possibili.

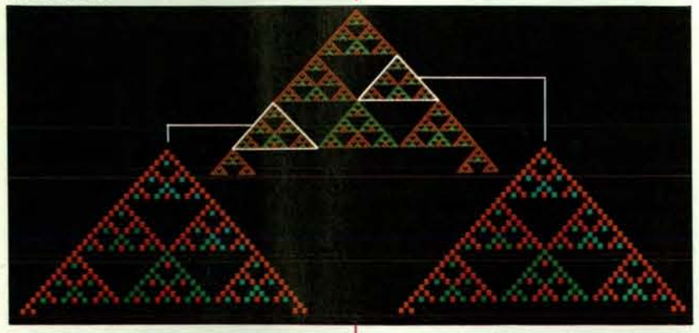
REGOLA



SIMULAZIONE



CONGETTURA



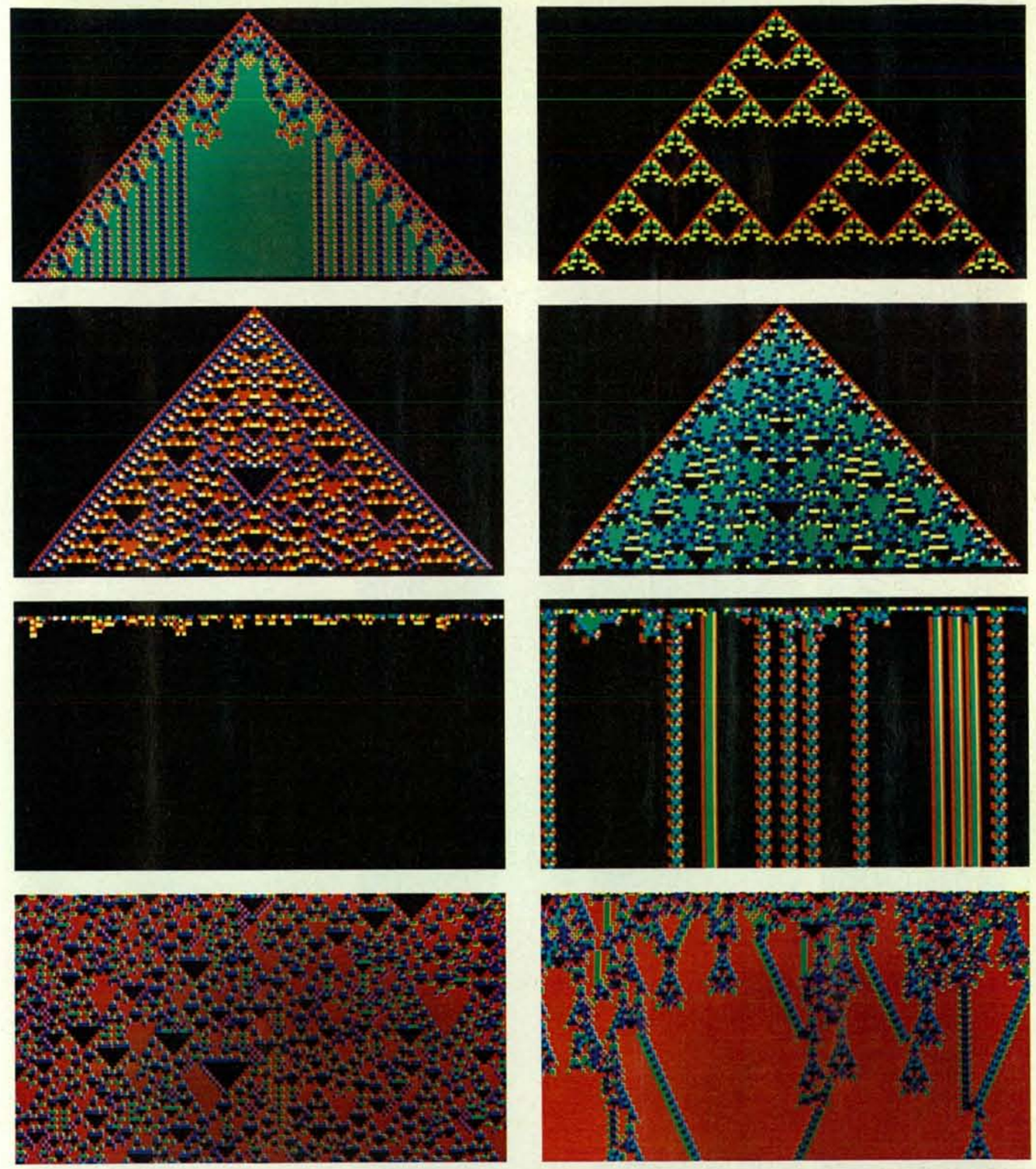
DIMOSTRAZIONE

La matematica sperimentale è una tecnica d'esplorazione resa in gran parte possibile dall'impiego dei calcolatori. Qualunque insieme di regole matematiche può essere applicato ripetutamente da un calcolatore e le conseguenze possono essere studiate in maniera sperimentale. Per esempio, per studiare una configurazione generata dall'automa cellulare definito dalla regola illustrata, si comincia simulando esplicitamente al calcolatore molti passi dell'evoluzione dell'automa cellulare. Osservando la configurazione ottenuta, si giunge a formulare la congettura che essa sia «frattale» cioè simile a se stessa, nel senso che le sue parti, ingrandite, hanno la stessa forma generale del tutto. Una volta che la congettura sia stata fatta, è abbastanza semplice dimostrarla con tecniche matematiche classiche. La dimostrazione può essere basata sul fatto che le condizioni iniziali della crescita a partire da certe cellule della configurazione sono le stesse che a partire dalla primissima cellula. Il numero dei risultati matematici ottenuti con esperimenti al calcolatore è in aumento e alcuni sono stati poi riottenuti con i metodi matematici tradizionali.

L'evoluzione di un automa cellulare a partire da una data configurazione iniziale può essere considerata come una computazione che elabora le informazioni contenute nella configurazione. Per gli automi cellulari che manifestano un comportamento semplice, questa computazione è semplice; per esempio può servire soltanto per individuare le successioni di tre cellule consecutive che abbiano 1 come valore iniziale. D'altra parte, può darsi che l'evoluzione degli automi cellulari che manifestano un comportamento complicato corrisponda a una computazione complicata.

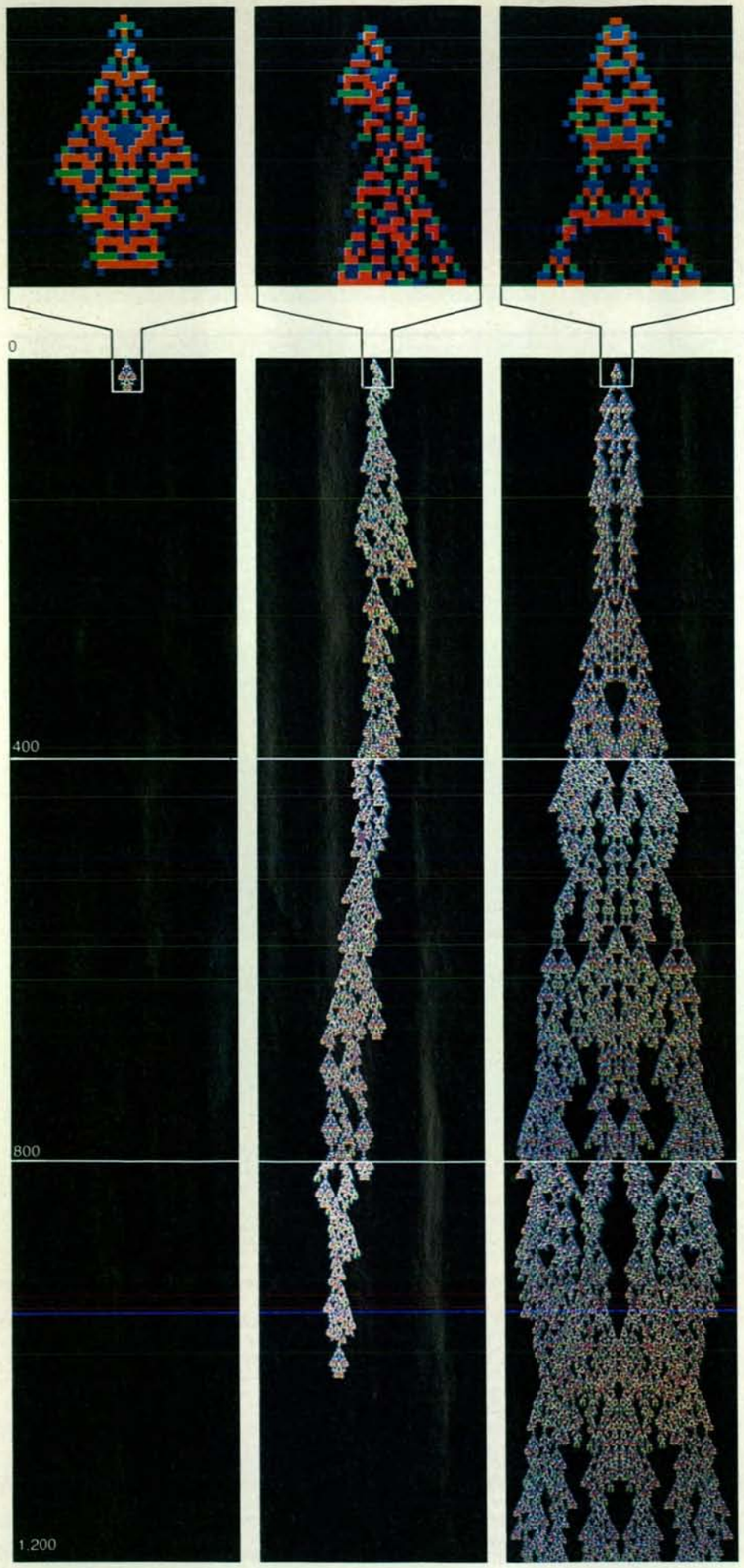
Tramite una simulazione esplicita di ciascun passo, è sempre possibile stabilire l'esito di un dato numero di passi nell'evoluzione di un automa cellulare. Il problema è se vi possa o no essere una procedura più efficiente: può esistere una scorciatoia della simulazione passo passo, cioè un algoritmo che trovi l'esito dell'evoluzione di un automa cellulare cui si giunge dopo molti passi, senza dover svolgere effettivamente tutti i passaggi? Un algoritmo siffatto potrebbe essere attuato da un calcolatore e prevederebbe l'evoluzione di un automa cellulare senza simularla esplicitamente. Il suo funzionamento sarebbe basato sulle circostanze che il calcolatore potrebbe svolgere una computazione più raffinata di quella consentita all'automa cellulare, e quindi ottenere lo stesso risultato con meno passaggi. Sarebbe come se l'automa cellulare dovesse calcolare sette per 18 trovando esplicitamente la somma di sette addendi uguali a 18, mentre il calcolatore è in grado di trovare lo stesso prodotto con il solito procedimento della moltiplicazione. Questa scorciatoia esiste solo se il calcolatore è in grado di svolgere un calcolo che sia intrinsecamente più avanzato del calcolo rappresentato dall'evoluzione dell'automa cellulare.

È possibile definire una certa classe di problemi, chiamati problemi computabili, che possono essere risolti in un tempo finito impiegando algoritmi ben definiti. Un calcolatore semplicissimo come una macchina addizionatrice può risolvere solo un esiguo sottoinsieme di questi problemi; esistono tuttavia calcolatori universali che possono risolvere qualunque problema computabile. Un calcolatore digitale reale è in pratica una macchina universale di questo genere. L'unità centrale di elaborazione del calcolatore può eseguire una gamma di istruzioni molto ampia, tanto che queste istruzioni possono costituire gli elementi per tradurre in un programma ed esprimere qualunque algoritmo. Si è dimostrato che, oltre al calcolatore digitale, vi sono molti altri sistemi in grado di eseguire computazioni universali. Tra di essi vi sono parecchi automi cellulari: per esempio si è dimostrato che la computazione universale è alla portata di un semplice automa cellulare bidimensionale avente 0 o 1 in ogni cellula. Vi sono fondate ragioni per ritenere che anche parecchi automi cellulari unidimensionali siano calcolatori universali; i candidati più semplici possono ave-



2213310_5	4200410_5	■ = 0
331240_5	2024310_5	■ = 1
1100400_5	2231000_5	■ = 2
131210_5	3211310_5	■ = 3
		■ = 4

Un comportamento complesso si può osservare anche in sistemi fatti di componenti semplici. Gli otto automi cellulari illustrati sono costituiti da file di cellule che assumono un valore fra cinque possibili. Il valore di ciascuna cellula è determinato da una semplice regola, basata sui valori delle sue vicine situate sulla fila precedente. Ciascuna configurazione è generata dalla regola il cui numero in codice è fornito nella legenda *(si veda l'illustrazione a pagina 149)*. Le configurazioni delle quattro fotografie in alto sono cresciute a partire da un'unica cellula in colore; perfino in questo caso le configurazioni generate possono essere complesse e avere un aspetto affatto casuale. Le configurazioni complesse che si formano in processi fisici come la turbolenza di un fluido possono ben nascere dallo stesso meccanismo. Le configurazioni complesse generate dagli automi cellulari possono servire anche come sorgenti di numeri realmente casuali ed essere applicate alla cifratura dei messaggi mediante la trasformazione di un testo in un crittogramma casuale. Le configurazioni delle quattro fotografie in basso cominciano con stati disordinati. Anche se i valori delle cellule di questi stati iniziali sono scelti a caso, l'evoluzione degli automi cellulari dà origine a strutture appartenenti a quattro classi fondamentali. Nelle due classi illustrate nella terza fila di fotografie il comportamento a lungo termine degli automi cellulari è abbastanza semplice; nelle due classi riportate nell'ultima riga il comportamento può essere molto complesso.

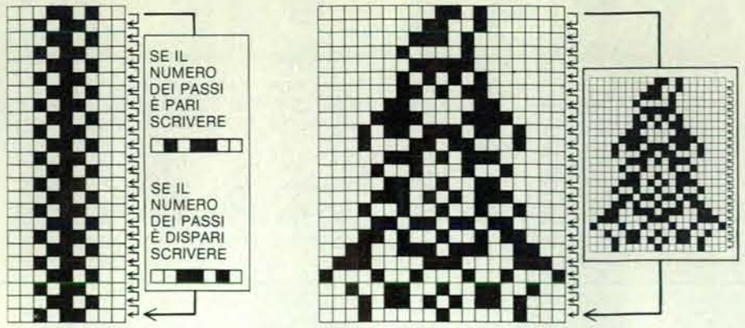


re in ciascuna cellula tre valori e le loro regole di evoluzione prendono in considerazione soltanto le cellule più vicine.

Gli automi cellulari capaci di svolgere la computazione universale possono imitare il comportamento di qualsiasi calcolatore; poiché qualunque processo fisico può essere rappresentato da un processo computazionale, essi possono imitare anche il comportamento di qualunque sistema fisico. Se esistesse un algoritmo che potesse calcolare il comportamento di questi automi cellulari in meno tempo di quanto ne occorre agli automi stessi per evolversi, questo algoritmo consentirebbe di accelerare qualunque computazione. Poiché questa conclusione porterebbe a una contraddizione logica, ne segue che non può esistere una scorciatoia generale per prevedere l'evoluzione di un automa cellulare arbitrario. Il calcolo che fornisce l'evoluzione è irriducibile: il suo risultato si può trovare effettivamente solo simulando esplicitamente l'evoluzione. Quindi la simulazione diretta è in effetti il metodo più efficiente per determinare il comportamento di certi automi cellulari. Non c'è modo di prevedere la loro evoluzione: si deve semplicemente aspettare che avvenga.

Ancora non si sa quanto sia diffuso il fenomeno dell'irriducibilità computazionale fra gli automi cellulari o più in generale tra i sistemi fisici. Tuttavia è chiaro che non è necessario che gli elementi di un sistema siano molto complicati affinché la sua evoluzione globale sia computazionalmente irriducibile. Può darsi che l'irriducibilità computazionale sia quasi sempre presente quando il comportamento del sistema si presenta complicato o caotico. Non si conoscono formule matematiche generali che descrivano il comportamento globale di sistemi siffatti, e magari queste formule non saranno mai scoperte. In tal caso la simulazione esplicita

Problemi indecidibili possono presentarsi nell'analisi matematica dei modelli di sistemi fisici. Si consideri per esempio il problema di stabilire se una configurazione generata dall'evoluzione di un automa cellulare prima o poi svanirà, talché tutte le cellule diventeranno nere. Le configurazioni generate dall'automa cellulare qui illustrato sono così complicate che l'unica impostazione generale possibile per risolvere il problema è quella di simulare esplicitamente l'evoluzione dell'automa cellulare. Si trova che la configurazione ottenuta dallo stato iniziale illustrato a sinistra svanisce dopo 16 passi soltanto. Lo stato iniziale al centro genera una configurazione che per svanire richiede 1016 passi. Lo stato iniziale a destra dà luogo a configurazioni il cui destino resta oscuro anche dopo che la simulazione si sia protratta per molte migliaia di passi. In generale nessuna simulazione che duri un numero fisso di passi può determinare con certezza il comportamento ultimo dell'automa cellulare. Di conseguenza il problema se una certa configurazione prima o poi svanisca o invece perduri viene detto formalmente indecidibile. L'automa cellulare qui illustrato segue una regola identificata dal numero di codice 3311100320_4.



L'irriducibilità computazionale è un fenomeno che sembra intervenire in molti sistemi fisici e matematici. Il comportamento di qualunque sistema si può trovare mediante una simulazione esplicita dei passi della sua evoluzione. Quando il sistema è abbastanza semplice, tuttavia, è sempre possibile trovare una scorciatoia di questa procedura; una volta assegnato lo stato iniziale del sistema, il suo stato a ogni passo successivo può essere trovato direttamente da una formula matematica. Per il sistema illustrato schematicamente a sinistra, la formula richiede solo che si trovi il resto della divisione per 2 del numero di passi dell'evoluzione. Un sistema di tal genere si chiama computazionalmente riducibile. Un sistema come quello illustrato schematicamente a destra, invece, ha un comportamento così complicato che in genere per descriverne l'evoluzione non si può assegnare nessuna scorciatoia. Un sistema di tal genere si chiama computazionalmente irriducibile e la sua evoluzione può essere determinata effettivamente solo mediante la simulazione esplicita di tutti i passi. Sembra probabile che molti sistemi fisici e matematici per i quali finora non si conosce nessuna descrizione semplice siano in realtà computazionalmente irriducibili. Per studiare questi sistemi l'unica strada praticabile è quella dell'esperimento, fisico o computazionale. Quando il livello di descrizione diviene computazionalmente irriducibile, cominciano a presentarsi anche i problemi indecidibili. Nel formulare una teoria questi problemi debbono essere evitati, proprio come in meccanica quantistica si debbono evitare le misurazioni simultanee di posizione e velocità di un elettrone (impossibili per il principio di indeterminazione).

mediante un esperimento al calcolatore sarebbe l'unico metodo d'indagine esistente.

Buona parte della scienza fisica si è tradizionalmente concentrata sullo studio dei fenomeni computazionalmente riducibili, per i quali si possono fornire descrizioni globali semplici. Nei sistemi fisici reali, tuttavia, può benissimo darsi che la riducibilità computazionale sia l'eccezione anziché la regola. La turbolenza dei fluidi è probabilmente uno fra i tanti esempi di irriducibilità computazionale. Nei sistemi biologici può darsi che l'irriducibilità computazionale sia ancora più diffusa: può darsi che per determinare la forma di un organismo biologico a partire dal suo codice genetico, non si possa, essenzialmente, fare altro che seguire ogni stadio del suo sviluppo. Quando si è in presenza dell'irriducibilità computazionale, si deve adottare una metodologia che dipende pesantemente dalla computazione.

Una delle conseguenze dell'irriducibilità computazionale è che esistono problemi relativi al comportamento ultimo di un sistema che si possono sì porre ma ai quali non si può dare una risposta di piena generalità tramite alcun procedimento matematico o computazionale finito: problemi siffatti debbono essere quindi considerati indecidibili. Un esempio di questi problemi è se nel corso dell'evoluzione di un automa cellulare una configurazione particolare prima o poi si dissolverà. Rispondere alla domanda per un numero di passi definito, diciamo 1000, è immediato: è sufficiente simulare 1000 passi dell'evoluzione dell'automa cellulare. Ma per dare una risposta per un numero arbitrario di passi, occorre simulare l'evoluzione dell'automa cellulare per un numero di passi potenzialmente infinito. Se l'automa cellulare è computazionalmente irriducibile, non esiste alcuna alternativa efficiente a questa simulazione diretta.

La possibilità che esistano problemi indecidibili nei modelli matematici dei sistemi fisici può essere considerata un effetto del teorema di Gödel sull'indecidibilità in matematica. Questo teorema, che fu dimostrato da Kurt Gödel nel 1931, afferma che in tutti i sistemi matematici, tranne quelli semplicissimi, vi possono essere proposizioni che non si possono né dimostrare né confutare mediante un procedimento matematico o logico finito. La dimostrazione di una data proposizione può richiedere un numero indefinitamente grande di passaggi logici. Anche proposizioni che possono essere enunciate in forma succinta possono richiedere una dimostrazione arbitrariamente lunga: in effetti vi sono molti teoremi matematici semplici le cui uniche dimostrazioni note sono lunghissime. Inoltre i casi che debbono essere vagliati per dimostrare o confutare certe congetture sono spesso complicatissimi. Nella teoria dei numeri, per esempio, vi sono molti casi in cui il minimo numero che gode di una certa proprietà è molto grande; spesso questo numero può essere trovato solo vagliando uno dopo l'altro tutti i numeri interi. Queste circostanze rendono il calcolatore uno strumento sempre più essenziale per molte ricerche matematiche.

L'irriducibilità computazionale comporta molte limitazioni fondamentali sulla portata delle teorie relative ai sistemi fisici. Di un sistema si possono costruire modelli a molti livelli, che vanno dalla simulazione del moto delle singole molecole fino alla risoluzione delle equazioni differenziali relative alle proprietà globali. L'irriducibilità computazionale comporta che esista un livello supremo al quale si possono costruire i modelli astratti: sopra quel livello si possono ottenere risultati solo ricorrendo alla simulazione esplicita.

Quando il livello di descrizione diviene computazionalmente irriducibile, cominciano a presentarsi anche i problemi indecidibili. Nel formulare una teoria questi problemi debbono essere evitati, proprio come in meccanica quantistica si debbono evitare le misurazioni simultanee della posizione e della velocità di un elettrone (che sono impossibili per il principio di indeterminazione). Ma anche qualora siano eliminati quei problemi, resta ancora la difficoltà pratica di dare risposta a problemi che in linea di principio possono essere risolti. Il grado di difficoltà dipende fortemente dalla natura degli oggetti coinvolti nella simulazione. Se l'unico modo di fare una previsione meteorologica fosse quello di simulare il moto di ciascuna molecola dell'atmosfera, non sarebbe possibile eseguire alcun calcolo pratico. Nondimeno è probabile che le caratteristiche del tempo che c'interessano si possano studiare considerando le interazioni di grandi volumi di atmosfera e quindi dovrebbe essere possibile compiere simulazioni utili.

L'efficienza con cui si può simulare un sistema computazionalmente irriducibile dipende dal grado di raffinatezza computazionale di ciascun passo della sua evoluzione. I passi dell'evoluzione del sistema possono essere simulati dalle istruzioni di un programma; quanto minore è il numero delle istruzioni occorrenti per riprodurre ciascun passo, tanto più efficiente è la simulazione. Le descrizioni a livello superiore dei sistemi fisici richiedono di solito passaggi più complicati, proprio come una singola istruzione espressa in un linguaggio di programmazione di livello superiore corrisponde a molte istruzioni in un linguaggio di livello inferiore. Un singolo passo temporale di un'approssimazione numerica dell'equazione differenziale che descrive un getto di gas richiede una computazione più complicata di quella occorrente per seguire la collisione tra due molecole del gas. Viceversa ciascun passo della descrizione a livello superiore fornita dall'equazione differenziale corrisponde a un numero immenso di passaggi nella descrizione, a livello inferiore, degli urti molecolari. Il guadagno di efficienza che ne risulta compensa largamente il fatto che i passi singoli sono più complicati.

In generale l'efficienza di una simulazione aumenta via via che i livelli di descrizione diventano più elevati, finché le operazioni necessarie alla descrizione di livello superiore non si attagliano con le operazioni eseguite direttamente dal calcolatore che compie la simulazione. L'efficienza massima si ottiene quando il calcolatore raggiunge il massimo grado di analogia rispetto al sistema che viene simulato.

Vi è una differenza importante tra la maggior parte dei calcolatori esistenti e i sistemi fisici o i loro modelli: i calcolatori elaborano l'informazione serialmente, mentre i sistemi fisici la elaborano in parallelo. In un sistema fisico rappresentato da un automa cellulare, i valori delle cellule sono aggiornati tutti insieme a ogni passo temporale; in un programma di calcolo ordinario, invece, la simulazione dell'automa cellulare è compiuta mediante un'iterazione che aggiorna i valori delle cellule uno dopo l'altro. In tal caso è immediato scrivere un programma di calcolo che svolga un procedimento essenzialmente parallelo con un algoritmo seriale. Esiste ormai un'intelaiatura ben consolidata entro la quale si possono descrivere gli algoritmi per l'elaborazione in serie dell'informazione; molti sistemi fisici, d'altra parte, sembrano richiedere descrizioni che sono fondamentalmente di natura parallela. Non esiste ancora un'intelaiatura generale per l'elaborazione in parallelo, ma quando sarà costituita dovrebbe essere possibile fornire, dei fenomeni fisici, descrizioni ad alto livello più efficaci.

L'introduzione del calcolatore nella scienza è piuttosto recente, eppure la computazione costituisce già un'impostazione nuova per molti problemi. Essa rende possibile studiare fenomeni molto più complessi di quelli che potevano essere presi in considerazione prima e sta modificando l'orientamento e l'importanza di molte discipline scientifiche. La cosa forse più importante è che essa introduce nella scienza un nuovo modo di pensare; le leggi scientifiche cominciano a essere considerate come algoritmi e molte di esse vengono studiate tramite esperimenti al calcolatore. I sistemi fisici vengono considerati come sistemi computazionali che elaborano l'informazione in modo molto simile ai calcolatori. Sono stati resi accessibili all'indagine nuovi aspetti dei fenomeni naturali: è nato un paradigma nuovo.

GENIUS

VIVERE LA CIVILTÀ ELETTRONICA — SCOPRIRE LA NUOVA INTELLIGENZA

SOMMARIO DEL SECONDO NUMERO.

I GRANDI SERVIZI
3000 anni dopo, i giapponesi ridanno un volto ai Faraoni.

LE OPINIONI
Piero Angela: Una lezione dall'Olanda.
Luciano Lama: Il robot è contro il sindacato?

LE INCHIESTE
L'industria marchigiana sfida il Giappone.

LE SCOPERTE
Entriamo nei palazzi intelligenti.

LE TESTIMONIANZE
Un ministro italiano in viaggio nel futuro USA.

LO SPETTACOLO
Broadway: I pompieri contro il nuovo teatro italiano.

E ancora: La scienza, gli italiani e il caos - I giochi del Club dei cervelli - Un americano giudica l'Italia elettronica - Un sondaggio del CNR su studenti e computer in Italia - Ultimissime sul cervello - Che c'è di nuovo per la salute, per lo sport, per l'ufficio.

E tra le firme: Silvio Ceccato, Edward Cornish, Luciano Francesconi, Luigi Granelli, Isabella Lattes Coifmann, Menotti Cossu, Vittorio Merloni, Sabatino Moscati, Giorgio Nebbia, Alberto Oliverio.

(RI)CREAZIONI AL CALCOLATORE

di A. K. Dewdney

I fallimenti di un occhio digitale indicano che non ci può essere visione senza comprensione

Immaginate una scatola nera simile a una macchina fotografica. Davanti porta una lente, su un fianco un quadrante con varie posizioni come «albero», «casa», «gatto» e così via. Con il quadrante sulla posizione «gatto», andiamo a fare una passeggiata. Incontriamo un gatto seduto sulla veranda di un vicino. Se la scatola viene puntata sul gatto, si accende una luce rossa; se è puntata su qualcos'altro la luce rimane spenta.

Dentro alla scatola c'è una rétina digitale che invia impulsi a una rete logica a due strati: un esemplare del congegno chiamato perceptrone. Un tempo si sperava che i perceptroni sarebbero stati alla fine in grado di riconoscere oggetti del mondo reale, come nella storia che ho appena raccontato. Qualcosa, però, non andò nel verso giusto.

Gli anni cinquanta e sessanta furono anni di enorme creatività e sperimentazione nel campo, di recente sviluppo, della scienza dei calcolatori. Molti scienziati furono influenzati da paradigmi romantici come i sistemi ad autoorganizzazione, le macchine che apprendono e i calcolatori intelligenti, tanto che sono tentato di chiamare quel periodo l'età della cibernetica. Macchine incredibili che potevano vedere o pensare o addirittura riprodursi sembravano giusto dietro l'angolo. La più semplice era il perceptrone.

Un perceptrone è composto da una retina a forma di griglia suddivisa in cellule che ricevono luce. Come certe cellule della retina umana, ogni cellula del perceptrone si attiva se riceve abbastanza luce; altrimenti rimane spenta. È quindi ragionevole pensare l'immagine analizzata da un perceptrone come una griglia di caselle chiare e scure, come nell'illustrazione della pagina a fronte.

Oltre alla retina, in un perceptrone vi sono numerosissimi elementi primitivi in grado di prendere decisioni, che chiamerò demoni locali. Ogni demone locale esamina un sottoinsieme prefissato di cellule retiniche e riferisce le condizioni trovate a un'entità più complessa, dotata della capacità di prendere decisioni, che potrei chiamare demone capo. Più specificamente, ogni demone locale è dotato di un taccuino su cui sono elencate certe configurazioni che deve cercare nel suo «circondario», il sottoinsieme di cellule retiniche sotto la sua giurisdizione. Se appare una delle configurazioni elencate, il demone locale invia un segnale al demone capo; in caso contrario rimane in silenzio. Il compito del demone capo è più complesso, in quanto deve fare qualche calcolo aritmetico. Ogni segnale inviato da un demone locale è moltiplicato per un particolare intero positivo o negativo (il «peso» assegnato a quel demone locale) e i numeri risultanti vengono sommati. Se la somma è maggiore o uguale a una certa soglia prefissata, il demone capo dice sì; altrimenti dice no. Per evitare ipotesi sull'aspetto dei vari demoni, li ho raffigurati come scatole.

Ai demoni sono spesso assegnati compiti pericolosi o addirittura impossibili, come aprire minuscole porticine nella parete di un contenitore per lasciar passare molecole. Al confronto, i demoni del perceptrone hanno compiti facili. In effetti, i demoni locali potrebbero essere sostituiti da semplici circuiti logici e il lavoro del demone capo potrebbe essere svolto agevolmente da pochi registri, un sommatore e un comparatore (elementi dell'unità centrale di elaborazione di qualsiasi calcolatore). I demoni, però, hanno un fascino romantico con cui i congegni elettronici non possono competere.

Il compito del perceptrone è dire sì quando gli vengono presentate determinate configurazioni e dire no a tutte le altre. Delle prime configurazioni si dice che vengono riconosciute dal perceptrone. Anche se è molto dubbio che possa mai essere costruito un perceptrone che riconosca i gatti, altri tipi di riconoscimento sono ottenibili.

Un perceptrone può essere in certo modo programmato a riconoscere una data classe di configurazioni calibrando i pesi e la soglia. Ai demoni locali che forniscono testimonianze in favore di quella classe vengono assegnati pesi positivi, mentre a quelli che forniscono testimonianze contrarie vengono assegnati pesi negativi. La grandezza di ogni peso riflette l'importanza o il valore della testimonianza. Anche se i perceptroni analizzati in questa sede operano con un insieme prefissato di pesi, la nozione di programmazione gioca un ruolo centrale nella teoria dei pereeptroni sviluppata negli anni cinquanta.

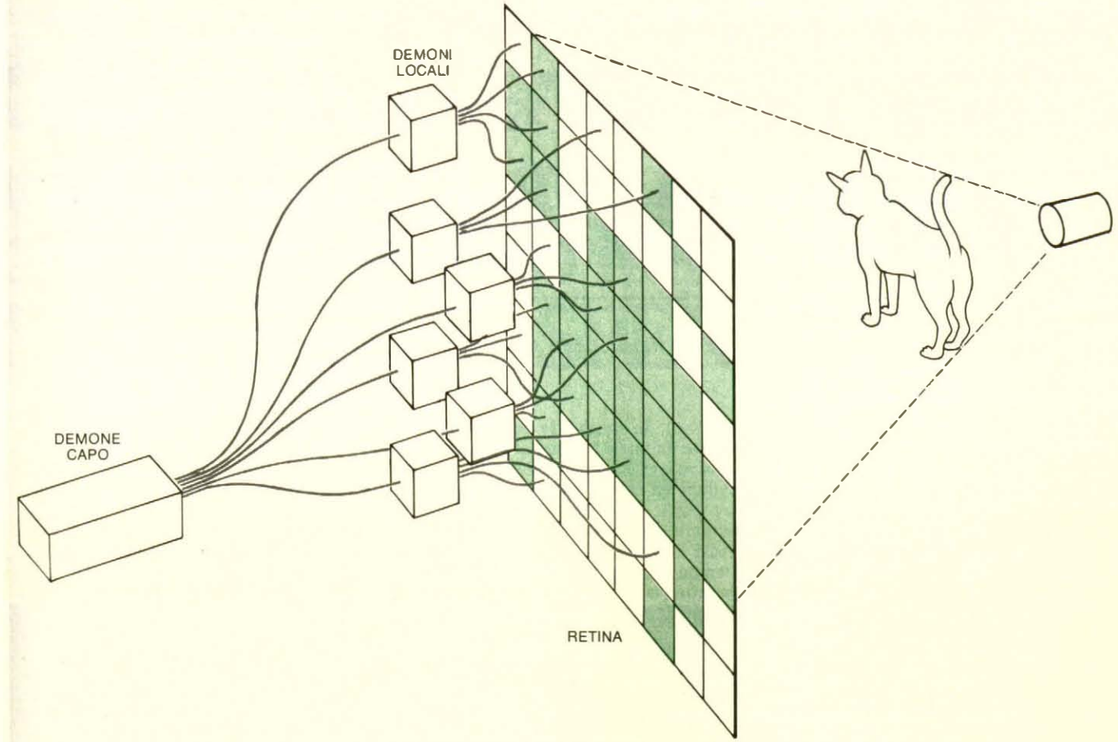
Il perceptrone che andiamo a descrivere riconosce un rettangolo scuro di qualsiasi dimensione o forma, posto in un punto qualsiasi della sua retina; riconosce anzi un numero qualsiasi di rettangoli del genere (anche zero), sempre che non ce ne siano due che si tocchino lungo un lato o in un angolo. Vi sono tre stadi nella costruzione del perceptrone. Prima di tutto, si pone un demone locale in ogni circondario costituito da 2×2 celle della retina. Si mettono poi tutte le sottoconfigurazioni dell'elenco *P (si veda la figura in alto di pagina 178)* sull'elenco di ogni demone locale. Per terzo, si assegna a tutti i pesi del demone capo il valore $+1$ e alla soglia il valore d, il numero dei demoni locali.

Questo progetto richiede un numero abbastanza limitato di demoni: se il perceptrone ha una retina $n \times n$, ci saranno $(n-1)^2$ demoni locali. A tutti sono assegnati pesi positivi, indicando così che tutti forniscono testimonianze positive per il riconoscimento di rettangoli. Per esempio, è facile vedere che quando sulla retina del perceptrone è proiettato un unico rettangolo, ogni insieme 2×2 di cellule deve contenere una delle sottoconfigurazioni dell'elenco *P*. Ne segue che ogni demone locale invia un segnale al demone capo e la somma ponderata dei segnali è naturalmente uguale a d. Il demone capo dice sì. D'altra parte, se una delle forme scure non è un rettangolo o se due rettangoli si toccano, allora almeno uno degli insiemi 2×2 contiene una sottoconfigurazione dell'elenco *N* riportato nella figura in alto di pagina 178. Almeno uno dei demoni locali, quindi, non trasmette alcuna segnalazione e il demone capo, calcolata una somma non superiore a $d-1$, dice no.

Si potrebbe ideare un perceptrone equivalente in cui ogni demone locale utilizza l'elenco più corto *N*. In questo caso tutti i pesi sarebbero -1 e la soglia sarebbe zero. Ogni demone locale fornirebbe testimonianza negativa nei confronti di una configurazione di rettangoli e il demone capo direbbe sì solo se nessuno dei demoni locali avesse inviato un segnale.

Il tipo di perceptrone definito sopra ha molte proprietà interessanti e vale la pena di dargli un nome. Senza specificare quale elenco di sottoconfigurazioni usino tutti i demoni locali, un congegno di questo genere sarà chiamato perceptrone a finestre, perché ogni demone locale guarda alla configurazione in ingresso attraverso una finestra 2×2. Per una retina $n \times n$ vi sono $(n-1)^2$ demoni e la soglia è uguale a questo numero.

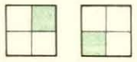
Generalmente parlando, i perceptroni sembrano dare il meglio di sé nel riconoscimento di figure geometriche. I perceptroni a finestre possono riconoscere non solo rettangoli ma anche «buchi neri» (cellule scure isolate), linee verticali e orizzontali, scale, scacchiere e molte altre configurazioni. Dipende tutto da quale insieme di sottoconfigurazioni 2×2 viene scelto per gli elenchi dei demoni locali *(si veda l'illustrazione a pagina 180)*. Ogni sottoinsieme delle 16 possibili sottoconfigurazioni 2×2 definisce un differente



Un perceptrone cerca di riconoscere un gatto

pereeptrone a finestre e ognuno dei 65 536 perceptroni a finestre risultanti riconosce una certa classe di configurazioni. Oppure no?

Il perceptrone a finestre basato sulle due sottoconfigurazioni riprodotte qui di seguito non riconosce nulla.



La ragione è semplice. Supposto d'avere una retina abbastanza larga, si scelga, in un punto al centro della retina, una finestra 2×2. Se la finestra contiene la prima delle due precedenti sottoconfigurazioni, si esamini la finestra una cellula più a destra: avrà una cellula scura nell'angolo a sinistra in alto e quindi il demone locale preposto a quel circondario non manderà alcun segnale al demone capo. Si ricordi che in un perceptrone a finestre *tutti* i demoni locali devono trasmettere una segnalazione, perché sia riconosciuta una configurazione. Se è presente la seconda sottoconfigurazione, lo spostamento della finestra di una cellula a sinistra porta a un'analoga contraddizione.

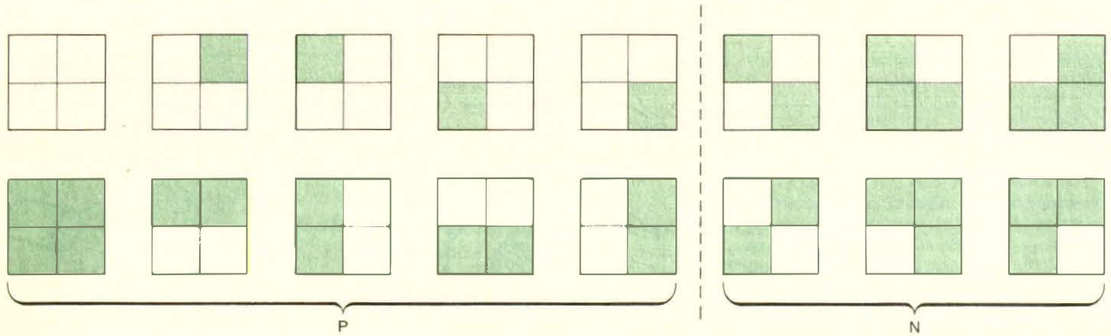
Quali sottoinsiemi delle 16 sottoconfigurazioni danno luogo a perceptroni a finestre che riconoscono davvero qualcosa? Il problema è probabilmente di difficile soluzione, ma illustra molto bene il tipo di domande che si potrebbero porre uno studioso di scienze dei calcolatori o un matematico interessato a questo argomento di fronte al fenomeno di un perceptrone che non riconosce nulla. Dato il grande numero di questi perceptroni, sarebbe meglio che la risposta prendesse la forma di un criterio o di un test di facile applicazione: dato un sottoinsieme di configurazioni 2×2, si applica il criterio e si ottiene una risposta per quel particolare sottoinsieme.

Non bisogna essere per forza dei professionisti nella teoria dei calcolatori per risolvere problemi di questo genere. Anche se non rientrano nel tipo di rompicapo normalmente presentati nelle rubriche di giochi matematici, richiedono lo stesso tipo di applicazione intellettuale. I lettori che abbiano risolto almeno uno dei rompicapo proposti da Martin Gardner nella rubrica di *Giochi matematici* de «Le Scienze» dovrebbero poter fare dei passi avanti nella soluzione di questo problema. Nella ricerca teorica, come nella scienza sperimentale, una risposta parziale è meglio di nessuna risposta.

Pionicre del lavoro sui perceptroni fu, negli anni cinquanta, Frank Rosenblatt della Cornell University. Rosenblatt e i suoi collaboratori, sia della Cornell sia di altri istituti, raggiunsero un certo ottimismo sulle prospettive dei perceptroni come dispositivi utili per il riconoscimento di configurazioni. Il «teorema di convergenza» diceva loro che in linea di principio i perceptroni potevano imparare a riconoscere configurazioni sottoponendo a controllo automatico i pesi usati dal demone capo. Il teorema afferma che qualsiasi calibratura dei pesi nella direzione di un miglior potere di riconoscimento può servire da base per ulteriori miglioramenti. Furono effettivamente costruiti perceptroni che, in alcune prove su semplici configurazioni, raggiunsero alti livelli di riconoscimento.

Quello che allora apparve un incoraggiante progresso era, in un certo senso, illusorio. Secondo Marvin L. Minsky e Seymour Papert, del Massachusetts Institute of Technology, gli entusiasti del perceptrone erano stati tratti in inganno dalla semplicità e dall'apparente successo dei loro congegni. Sotto la superficie stanno alcuni gravi difetti concettuali. Nel 1969 Minsky e Papert pubblicarono *Perceptrons*, un efficace libro che metteva in crisi l'idea di perceptrone evidenziando (con dimostrazioni) numerose cose che i perceptroni non potevano fare.

Una delle più rilevanti fra le carenze scoperte da Minsky e Papert era l'incapacità di certi perceptroni di riconoscere quando una figura è connessa (cioè è tutta di un pezzo). Presupponendo che ogni demone locale ispezioni solo una regione limitata, Minsky e Papert fornivano esempi di quattro configurazioni fatte in modo che una di esse metta sempre in difficoltà un perceptrone il cui compito sia quello di riconoscere la connessione. Nella figura in basso della pagina seguente si possono vedere tali



Le sottoconfigurazioni 2 × 2 riconosciute da demoni locali positivi (P) *e negativi* (N)

configurazioni. Due di esse (*b* e *c*) sono figure connesse e le altre due (*a* e *d*) no.

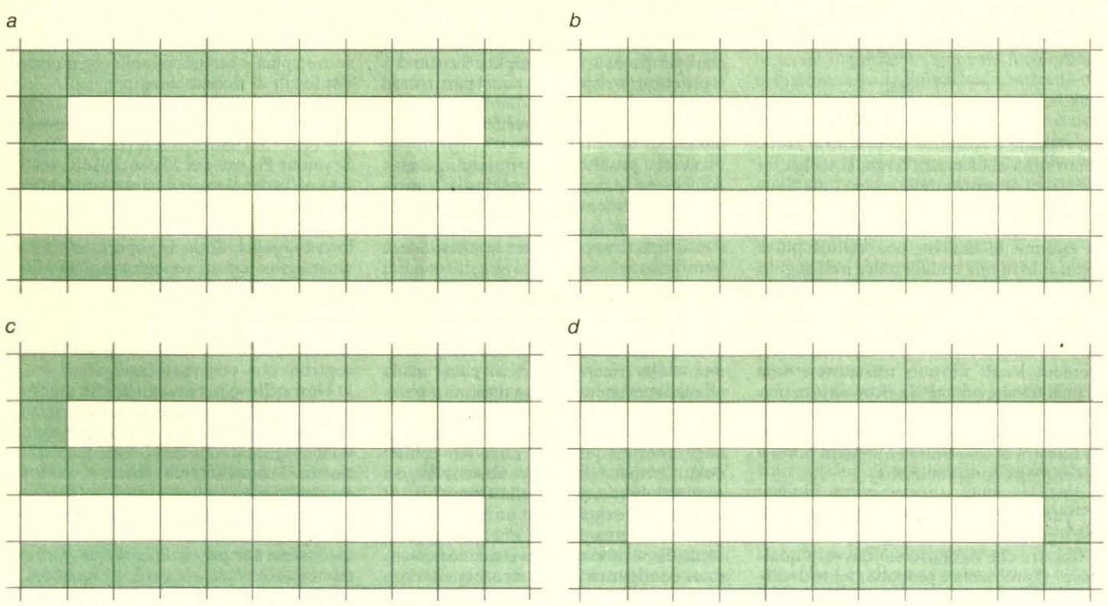
Supponiamo che qualcuno affermi di aver progettato un perceptrone, limitato nel diametro, capace di distinguere tra configurazioni connesse e non connesse. Per limitato nel diametro intendo che per qualche numero *m* ogni demone locale possa esaminare solo le caselle all'interno di una finestra $m \times m$. Per verificare l'affermazione, Minsky e Papert preparerebbero versioni delle loro quattro configurazioni fatte in modo da essere lunghe più di *m* cellule. I demoni locali possono allora essere classificati in tre insiemi disgiunti: i Demoni Sinistri esaminano almeno una cellula del lato sinistro della figura; i Demoni Destri esaminano almeno una cellula del lato destro; i Demoni Altri non sono né Sinistri né Destri.

Quando al perceptrone per la connessione viene presentata la figura *a*, o si sbaglia (e dice sì) oppure ha successo (e dice no). Se sbaglia, la verifica è ovviamente finita. Se ha successo, il passo successivo consiste nell'esaminare la somma sviluppata dal demone capo e dividerla in tre parti: *S*, *A* e *D*, rappresentanti le somme ponderate dei demoni Sinistri, Altri e Destri che inviano un segnale al demone capo quando viene proiettata sulla retina la configurazione *a*. Dato che il perceptrone per la connessione dice no, la somma di *S*, *A* e *D* non raggiunge la soglia. Se ora si sostituisce la configurazione *a* con la configurazione *b*, solo i Demoni Sinistri cambiano la loro risposta, visto che cambiano solo le cellule lungo il lato sinistro della figura. Supponiamo che la somma parziale *S* divenga *S'*. Invece, se si sostituisce la configurazione *a* con la configurazione *c*, cambiano solo le cellule lungo il lato destro e solo i Demoni Destri cambiano risposta, diciamo da *D* a *D'*.

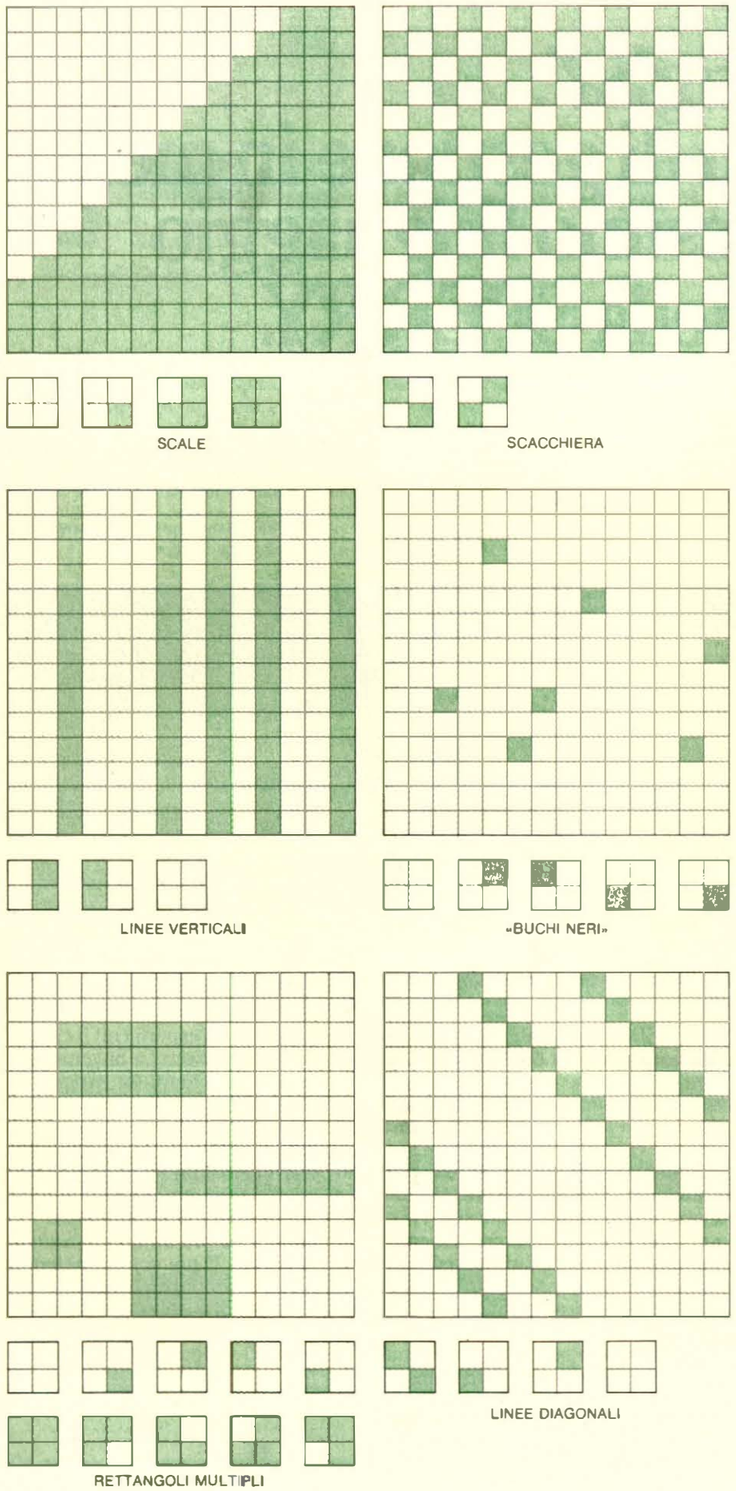
A questo punto il perceptrone si trova in una posizione molto curiosa. Dato che *b* e *c* sono connesse, deve rispondere sì in entrambi i casi e quindi le somme $S' + A + D$ e $S + A + D'$ devono almeno raggiungere la soglia. Sappiamo già, però, che $S + A + D$ è inferiore alla soglia perché *a* non era connessa. Ne segue che *S'* è maggiore di *S* e *D'* è maggiore di *D*. Il colpo di grazia viene quando il perceptrone si trova davanti la configurazione *d*. Qui sia le cellule di sinistra sia le cellule di destra della figura hanno cambiato stato rispetto alla configurazione *a* e il demone capo si trova a calcolare la somma $S' + A + D'$, che è certamente maggiore della soglia. Il demone capo dice sì e sbaglia.

Tra le altre carenze dei perceptroni scoperte da Minsky e Papert ci sono il numero assurdamente grande di demoni locali necessari per alcuni compiti di riconoscimento e il basso ritmo di apprendimento (o di convergenza) per altri compiti.

Non c'è forse da sorprendersi se in molti casi i perceptroni falliscono là dove il sistema visivo dell'uomo ha successo.



Quattro figure ideate per confondere un perceptrone per la connessione



Alcune configurazioni riconosciute da perceptroni a finestre

Ho rilevato in precedenza che i demoni locali e il demone capo potrebbero essere sostituiti da semplici circuiti di calcolo. Potrebbero venire sostituiti anche dai neuroni formali descritti per la prima volta negli anni quaranta da Warren S. McCulloch e Walter H. Pitts nel loro ormai classico lavoro sulle reti di neuroni. Questi neuroni formali sono molto più semplici dei neuroni umani; proprio come la complessità di un perceptrone organizzato come rete di neuroni a due strati non si avvicina alla complessità dei primi due strati della corteccia visiva umana. Per di più, «dietro» la corteccia visiva, per così dire, c'è un incredibile apparato analitico quasi completamente sconosciuto - qualcosa che manca del tutto nel modello della visione del perceptrone. Solo per cominciare a riprodurre questa maggiore complessità si dovrebbe sostituire il demone capo con una macchina di Turing, ma qui si sprofonda in un mare inesplorato di congetture e non procederò oltre.

Pur essendo occhi senza mente, i perceptroni hanno una certa semplicità che affascina e, almeno per alcune configurazioni, possiedono ben definiti poteri di riconoscimento. Mi chiedo se i lettori potrebbero scoprire altre configurazioni che rientrino nella competenza del perceptrone a finestre. Chi volesse affrontare il più arduo problema di quali sottoinsiemi delle 16 sottoconfigurazioni 2 × 2 portino a «buoni» perceptroni a finestre (quelli che riconoscono almeno una configurazione) troverà la questione leggermente semplificata se aggiunge una limitazione: le configurazioni riconosciute dovrebbero essere «trasportabili» - si dovrebbe cioè poterle spostare sulla retina senza modificare il fatto che esse vengano riconosciute. Questo requisito non solo esclude certi perceptroni a finestre ipersperializzati (per esempio quello che riconosce una singola cellula scura nell'angolo a destra in alto della sua retina) ma riflette anche l'idea che il perceptrone osserva una scena reale che si muove attraverso la retina, così come la esplora la mia immaginaria scatola nera.

Anche se i perceptroni a diametro limitato sono incapaci di distinguere le figure connesse da quelle non connesse, può essere possibile riconoscere la connessione in certe classi di figure. Per esempio, nella classe delle configurazioni formate da più rettangoli le figure connesse sarebbero quelle che comprendono esattamente un rettangolo. Riuscite a ideare un perceptrone che riconosca proprio queste configurazioni? I vostri demoni locali devono usare finestre 2 × 2, ma se è necessario potete assumere anche dei demoni aggiuntivi.

Da quanto detto in precedenza si poteva dedurre che la ricerca sui perceptroni avesse avuto termine dopo la pubblicazione di *Perceptrons* da parte di Minsky e Papert. Questo è vero nel senso che da allora non è più possibile un atteggiamento di totale fiducia nei perceptroni e nel loro potere di riconoscimento. D'altra parte, non era affatto nelle intenzioni di Minsky e Papert porre termine alla ricerca nel campo della teoria dei perceptroni.

I limiti precisi delle possibilità di questi congegni, semplici ma a volte efficaci, sono ancora da scoprire.

Rosenblatt, il cui lavoro toccava anche la psicologia e la neurobiologia, morì in un tragico incidente di barca il giorno del suo quarantatreesimo compleanno, l'11 luglio 1971, nel Maryland.

L'articolo sui congegni analogici ha avuto una gratificante risposta da parte dei lettori: sono stati suggeriti non meno di 17 nuovi congegni e sono state proposte tre soluzioni corrette al problema della luce nella scatola a specchi.

Prima di entrare in argomento, però, devo correggere un errore. Il più veloce algoritmo digitale a me noto per trovare l'involucro convesso di un insieme di punti nel piano richiede nell'ordine di $n \log n$ operazioni, non di $n \log \log n$. Il congegno analogico a elastico che risolve lo stesso problema venne inventato nel 1957 da George J. Minty, Jr., dell'Indiana University. Minty fa anche notare che la tecnica della pellicola di sapone per trovare alberi di Steiner minimi fu proposta da William Wiehle nel 1958.

Il congegno a laser per scoprire se un numero n è primo è stato criticato da David Zimmerman di Beaver Dam, Wisconsin. Egli nota che la luce deve essere riflessa n volte nel tragitto dal laser al rilevatore e, dato che la velocità della luce è finita, il tempo di soluzione è proporzionale a n. Se la dimensione del problema è definita come il numero di cifre di n, il tempo di soluzione cresce in modo esponenziale e il congegno non è più veloce di un algoritmo digitale.

Il fatto che la velocità della luce sia finita ha dato da pensare anche a Steven P. Hendrix, di New Braunfels, Texas, il quale ha osservato che forse si dovrebbe attendere un tempo lunghissimo per veder emergere la luce. Io chiedevo quale proprietà del percorso della luce è misurata dalla scatola a specchi e Hendrix è stato tra quelli che hanno risolto il problema, notando che chiedersi se la luce emerga è equivalente a chiedersi se una linea retta nel piano interseca un punto con coordinate intere.

Si immagini un frutteto infinito con alberi infinitamente sottili piantati a griglia quadrata. Se si spara una pallottola da un albero in una direzione arbitraria, la pallottola colpirà mai un altro albero? Lo farà se l'angolo, rispetto alle file di alberi, ha un'inclinazione razionale. Se l'albero colpito è p file a nord e q file a est del punto da cui si è fatto fuoco, l'inclinazione è p/q. Gli specchi nella scatola fanno solo piegare il percorso della pallottola.

L'unico modo per rendere giustizia al gran numero di congegni descritti dai lettori sarà quello di dedicarvi un intero numero della rubrica; lo farò nei prossimi mesi. Nel frattempo, voglio almeno citare alcuni dei congegni più interessanti.

Peter F. Ash, della St. Joseph's University, ha proposto un modo per risolvere un'equazione cubica consistente nell'immergere dei solidi in una cisterna d'acqua. Tom Digby di Los Angeles ha osservato che la potenza di calcolo di un congegno analogico può essere attribuita alla sua capacità di compiere molti processi in parallelo. Ha anche mostrato come predisporre n calcolatori digitali per ordinare n numeri in un tempo lineare, eguagliando la prestazione del congegno analogico a spaghetti.

Eric Halsey, dell'Università di Washington, ha inviato la descrizione di un congegno per il percorso più lungo fatto di «serpenti». Ogni spigolo del grafo è rappresentato da un elastico fatto passare attraverso un numero intero di grani. Risulta il percorso più lungo quando il congegno viene tirato e poi liberato? Un altro dei suoi congegni misura la lunghezza del percorso più breve tra due vertici di un grafo. Ogni lato è un pezzo di miccia e nel secondo vertice c'è un petardo; si accendono le micce nel primo vertice e ci si tira indietro: il tempo necessario perché esploda il petardo è proporzionale alla lunghezza del percorso più breve.

Palmer O. Hansen, Jr., di Largo, Florida, mi ha ricordato che il planimetro, un dispositivo meccanico per misurare l'area, potrebbe essere considerato un congegno analogico. Dale T. Hoffman, del Bellevue Community College di Washington, ha individuato qualche altro problema risolvibile con la tecnica della pellicola di sapone, tra cui un intelligente calcolo della legge di Snell. David Kimball, di San Diego, risolve labirinti pompando acqua nel labirinto e seguendo la corrente per uscire. Un altro simpatico congegno è stato descritto da J. H. Lueth, della United States Metals Refining Company di Carteret, New Jersey. SLAG (Smelter-Location Analog Gadget) trova la localizzazione per un fonditore che minimizzi i costi del trasporto di calcare, carbone e minerale. Il problema è risolto con tre buchi in una scatola e tre pesi legati insieme con lo spago. Lo stesso congegno è stato citato anche da Hendrix.

Tony Mansfield, del British National Physical Laboratory di Teddington, risolve problemi di programmazione lineare con una struttura fatta di pezzi di un gioco di costruzioni. Thomas A. Reisner, dell'Università Laval del Quebec, genera una mappa isometrica di una superficie stendendo su di essa una zanzariera. Una forte luce proveniente dall'alto crea un disegno moiré per l'interferenza della rete con la propria ombra.

L'industria statunitense degli agrumi utilizza chiaramente un congegno analogico per selezionare la frutta. Le arance rotolano nel canale tra due tubi non del tutto paralleli e cadono quando la distanza tra i due tubi è uguale al diametro dell'arancia. John P. Schwenker, di Louisville, Colorado, ha trovato una volta il centro di gravità di un pezzo di apparecchiatura con una variante della tecnica del bilanciamento dei piatti ideata da Ronald L. Graham. Quando l'apparecchiatura è trascinata da una fune lungo una superficie liscia, il piano verticale passante per la fune passa anche attraverso il centro di gravità. L'intersezione di tre di questi piani identifica il centro di gravità stesso.